

**COPS  
MICROCONTROLLERS  
DATABOOK**

---

**NATIONAL  
SEMICONDUCTOR  
CORPORATION**



**COPS  
MICROCONTROLLERS  
DATABOOK**

---

**NATIONAL  
SEMICONDUCTOR  
CORPORATION**



<b>Introduction COPS Family</b>	<b>1</b>
<b>Single-Chip Microcontrollers</b>	<b>2</b>
<b>ROMless Microcontrollers</b>	<b>3</b>
<b>Piggyback Microcontrollers</b>	<b>4</b>
<b>MICROWIRE™ Peripherals</b>	<b>5</b>
<b>Standard Controllers</b>	<b>6</b>
<b>EPROMs and Support Circuits</b>	<b>7</b>
<b>Development Systems and User's Manuals</b>	<b>8</b>
<b>COPS Application</b>	<b>9</b>
<b>Appendix/Physical Dimensions</b>	<b>10</b>





### **LIFE SUPPORT POLICY**

**NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:**

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

TRI-STATE is a registered trademark of National Semiconductor Corp.

COPS, ISE, MICROBUS, MICROWIRE, STARPLEX and STARPLEX II are trademarks of National Semiconductor Corp.

# The COPS™ Family

The COPS Family of microcontrollers provides a flexible, cost-effective system solution in applications requiring timing, counting or other control functions. COPS can be used to replace discrete logic in high-volume consumer products and low-volume industrial products allowing you to add features, miniaturize and reduce component count.

All of the programmable microcontrollers in the COPS Family share a common architecture, pin-out and instruction set, so that once you have programmed one, you can design with the entire family. In addition, compatible standard peripherals and pre-programmed microcontrollers can add extra capability to your design at off-the-shelf prices.

National Semiconductor recognized the need for a family of controllers that would grow with our customers' needs. The COPS Family, with its programming and cost efficiency, versatility and ease of development is at the leading edge of technology. We are committed to keeping it there by continually phasing in new design concepts and fabrication methods. This systematic evolution brings you state-of-the-art devices to drive your products into the future.

COPS devices are produced on some of the largest fabrication lines in the semiconductor industry. Located around the world, these lines actually "second source" each other, ensuring you a steady supply of products when you need them. Availability, combined with the money that is saved by not having to retrain from one product to the next, has made the COPS Family a standard for many companies.

## **COPS: Cost-efficiency**

The COPS Family was designed with efficiency in mind. The more the controller can do, the greater are your product alternatives. Several approaches have been taken to allow you to add capability to your products while lowering costs:

*We've designed the industry's most ROM-efficient instruction set.* Every COPS microcontroller uses the same ROM-efficient instruction set, which often requires significantly less ROM to carry out a set of tasks than with other 4- or even 8-bit devices. As your program develops and you find that you require less (or more) ROM than you originally anticipated, you can easily go to other COPS devices—of larger or smaller ROM size—without starting over.

*Our dual CPUs are an economical alternative to bigger memories.* National is the first to develop an architecture that permits two CPUs to be placed onto a single device. Speed is increased because one CPU can process regular events while the other handles random tasks, eliminating the need to shuffle back and forth between diverse, time-critical operations. Since both CPUs access common memories, program efficiency is virtually doubled at little extra cost.

*Standard peripherals inexpensively add distributed processing and unique capabilities.* Two of these devices are of special interest for their ability to increase speed and reduce power requirements. The *COP452 Frequency/Counter* assists the processor in handling high-frequency information, increasing system speed by a factor of up to 100. The *COP498 RAT™ Chip* (CMOS RAM and Timer) allows the CPU to "sleep" and "wake up" under software control, reducing an NMOS controller's power consumption to a level approaching CMOS controllers at a much lower system cost. Both of these devices have other capabilities that are detailed in their respective data sheets.

*MICROWIRE™ makes efficient use of every I/O line.* The COPS Family is designed with National's MICRO-WIRE system, which permits serial data exchange with only three wires. This reduces I/O lines, enabling the use of a more cost-effective package (i.e., fewer number of pins) or the addition of more features and capability to your final product.

## **COPS: Design Flexibility**

Never before have so many options been available with a common architecture and pin-out. Once you choose the COPS Family, any of the following options can be selected or modified during the product development cycle:

- *Capacity.* Memories range from 0.5k ROM and 32 × 4-bit RAM to 4k ROM and 256 × 4-bit RAM. The 2k ROM-size devices are available with either single or dual CPUs.
- *Environment.* COPS circuits can be fabricated by the optimum process technology for any application, from high-speed NMOS to low-power NMOS to very low-power CMOS. And operating temperature ranges are available from -55°C to +125°C.
- *Mask-programmable options.* Several options can be masked onto COPS devices simultaneously with the user's program. They include up to four basic clock oscillators, as well as an array of I/O configurations (i.e., LED drive, open-drain and TRI-STATE® circuitry). In addition, COPS devices can serve as "smart" microprocessor peripherals by selecting the MICROBUS™ option, National's standard interconnect for 8-bit data transfer.

## **COPS: Development Ease**

The COPS Family places a variety of tools and professional support at your disposal to make designing easier. Several alternatives are available to you, depending on your in-house capabilities, product mix and marketing strategies. Regardless of which path you choose, National Semiconductor's field and factory applications specialists are available throughout the design process.

- *COP400 Product Development System (PDS)*. This powerful, easily understood programming system performs complex software development and debug tasks with a minimum of effort — and investment. You interact with the system via a teletype or CRT console and can attach a printer for fast program listings. Data is stored on a floppy diskette for fast, easy access and for convenience in providing National with the mask program. A real-time in-system emulator board allows you to develop and debug your COPST<sup>™</sup> device from within your hardware environment.

National's complete PDS training course will teach you how to develop all of your products with the COPS Family. So if your company needs to develop in-house design capabilities for a minimal capital outlay, PDS makes a lot of sense.

- *The COPS In-System Emulator (ISE<sup>™</sup>) Package* is for companies who already own, or are considering, a STARPLEX<sup>™</sup> Development System. A target board plugs directly into any STARPLEX or STARPLEX II<sup>™</sup> system, giving it virtually the same diskette storage and real-time emulation capabilities as the COP400 PDS. The powerful STARPLEX system also supports National's state-of-the-art programmable microprocessors, making it ideal if your company uses a wide

range of programmable products.

- *Prototyping*. ROMless or piggyback COPS devices can be interfaced with a standard PROM to facilitate development and debugging, particularly when pre-market testing is desirable prior to masking the final part. They can also provide an effective alternative to mask-programming in low-volume applications or when your competitive environment demands fast product modifications.
- *National's COPS Controller Engineering Group*. New companies, or those with little time or in-house design expertise, can take advantage of our Controller Engineering Group. These professionals will put their vast COPS programming and applications experience to work in implementing your specifications into a COPS-controlled system.

### **A Mutual Commitment**

National Semiconductor has committed extensive design and fabrication resources to providing you with a steady stream of cost-efficient, flexible, easily developed COPS devices. This data book will help familiarize you with the many alternatives that are currently available to help you bring your ideas to market.





# Table of Contents

## Section 1 Introduction COPS Family

COPSTM Microcontroller Family Guide .....	1-3
COPS ROMless Microcontroller Family Guide .....	1-3

## Section 2 Single-Chip Microcontrollers

COP410C/COP411C, COP310C/COP311C Fully Static, Single-Chip CMOS Microcontrollers .....	2-3
COP410L/COP411L, COP310L/COP311L Single-Chip N-Channel Microcontrollers .....	2-5
COP420/COP421/COP422, COP320/COP321/COP322 Single-Chip N-Channel Microcontrollers .....	2-23
COP420C/COP421C, COP320C/COP321C Single-Chip CMOS Microcontrollers .....	2-45
COP420L/COP421L/COP422L, COP320L/COP321L/COP322L Single-Chip N-Channel Microcontrollers .....	2-64
COP440/COP441/COP442, COP340/COP341/COP342 Single-Chip N-Channel Microcontrollers .....	2-88
COP444C/COP445C, COP344C/COP345C Single-Chip CMOS Microcontrollers .....	2-111
COP444L/COP445L, COP344L/COP345L Single-Chip N-Channel Microcontrollers .....	2-112
COP464 and COP484 Single-Chip 3k and 4k Microcontrollers .....	2-134
COP2440/COP2441/COP2442, COP2340/COP2341/COP2342 Single-Chip Dual CPU Microcontrollers .....	2-135

## Section 3 ROMless Microcontrollers

COP401L ROMless N-Channel Microcontroller .....	3-3
COP402/COP402M, COP302/COP302M ROMless N-Channel Microcontrollers .....	3-17
COP404/COP304 ROMless N-Channel Microcontrollers .....	3-36
COP404L/COP304L ROMless N-Channel Microcontrollers .....	3-46
COP2404/COP2304 ROMless Dual CPU Microcontrollers .....	3-65

## Section 4 Piggyback Microcontrollers

COP420R/COP444LR Piggyback-EPROM Microcontroller .....	4-3
COP440R/COP2440R Piggyback-EPROM Microcontroller .....	4-18



# Table of Contents (continued)

## Section 5 MICROWIRE™ Peripherals

COP431, COP432, COP434, COP438 8-Bit Serial I/O A/D Converters with Multiplexer Options.....	5-3
COP452/COP453, COP352/COP353 Frequency Generator and Counter.....	5-13
COP470, COP370 V.F. Display Driver.....	5-44
COP472 Liquid Crystal Display Controller.....	5-51
COP498/COP398 Low Power CMOS RAM and Timer (RAT™), COP499/COP399 Low Power CMOS Memory.....	5-58
DS8906 AM/FM Digital Phase-Locked Loop Synthesizer.....	5-70
DS8907 AM/FM Digital Phase-Locked Loop Frequency Synthesizer.....	5-76
DS8908 AM/FM Digital Phase-Locked Loop Frequency Synthesizer.....	5-82
MM5445, MM5446, MM5447, MM5448 VF Display Drivers.....	5-89
MM5450, MM5451 LED Display Drivers.....	5-92
MM5452, MM5453 Liquid Crystal Display Drivers.....	5-97
MM5480 LED Display Driver.....	5-102
MM5481 LED Display Driver.....	5-105
MM5484, MM5485 16-, 11-Segment LED Display Drivers.....	5-108
MM58201 Multiplexed LCD Driver.....	5-111
MM58248, MM58241 High Voltage Display Drivers.....	5-116
MM58348, MM58341 High Voltage Display Drivers.....	5-119

## Section 6 Standard Controllers

MM57409 Super Number Cruncher.....	6-3
MM57436 Decimal/Binary Up/Down Counter.....	6-6
MM57455 Advanced Educational Arithmetic Game.....	6-13
MM57459 8-Digit LED Direct-Drive Memory Calculator.....	6-16

## Section 7 EPROMS and Support Circuits

MM2716 16,384-Bit (2048 × 8) UV Erasable PROM.....	7-3
NMC27C16 16,384-Bit (2048 × 8) UV Erasable CMOS PROM.....	7-9
MM2758 8192-Bit (1024 × 8) UV Erasable PROM.....	7-15
MM54C373/MM74C373 TRI-STATE® Octal D-Type Latch MM54C374/MM74C374 TRI-STATE Octal D-Type Flip-Flop.....	7-21
DM54LS373/DM74LS373, DM54LS374/DM74LS374 Octal D-Type Transparent Latches and Edge-Triggered Flip-Flops.....	7-27



# Table of Contents (continued)

## Section 8 Development Systems and User's Manuals

COP400-PDS Product Development System	8-3
COP400 Product Development System User's Manual	8-5
COP400 In-System Emulator Boards User's Manual	8-113
STARPLEX™ Development System	8-134
STARPLEX II™ Development System	8-141
COPST™ In-System Emulator (ISET™) Package	8-157
STARPLEX COPS Cross-Assembler User's Manual	8-163
SPM-A15, SPM90/A15 COP400 Emulator User's Manual	8-240

## Section 9 COPS Application

COPST™ Family User's Guide	9-3
COP Note 1 Analog to Digital Conversion Techniques With COPST™ Family Microcontrollers	9-78
COP Note 2 COPST™ Television Controller	9-113
COP Note 3 Design Considerations for a COP420C-Based Telephone-Line Powered Repetory Dialer	9-118
COP Note 4 The COP444L Evaluation Device 444L-EVAL	9-123
COP Note 5 Oscillator Characteristics of COPST™ Microcontrollers	9-128
COP Note 6 Triac Control Using the COP400 Microcontroller Family	9-152
COP Note 7 Testing of COPST™ Chips	9-159
COP Brief 1 SIO Input/Output Register Description	9-166
COP Brief 2 Easy Logarithms for COP400	9-168
COP Brief 3 Use of Macro-Assembled Code	9-179
COP Brief 4 Bus Considerations	9-181
COP Brief 5 Software and Opcode Differences in the COP444L Instruction Set	9-182
COP Brief 6 RAM Keep-Alive	9-183
COP Brief 7 MICROBUS™ Programming Considerations	9-184
COP Brief 8 COPST™ Peripheral Chips	9-185
COP Brief 9 Serial Interface Between COPST™ Microcontrollers and Peripheral Chips	9-187
COP Brief 11 Power Seat with Memory	9-189
COP Brief 12 An Automotive Diagnostics Display	9-193
COP Brief 13 An Electronic Speedometer and Odometer with Permanent Mileage Accumulation	9-195
COP Brief 14 COP420C Voltage, Current, and Frequency	9-201

## Section 10 Appendix

COP420-HGZ/N Preprogrammed Single-Chip Microcontroller for Musical Organ	10-3
COP420L-HSB Digital Tuning System Controller	10-5
Ordering Information/Physical Dimensions	10-10





# Alphanumerical Index

COP302	ROMless N-Channel Microcontrollers	3-17
COP302M	ROMless N-Channel Microcontrollers	3-17
COP304	ROMless N-Channel Microcontrollers	3-36
COP304L	ROMless N-Channel Microcontrollers	3-46
COP310C	Fully Static, Single-Chip CMOS Microcontroller	2-3
COP310L	Single-Chip N-Channel Microcontroller	2-5
COP311C	Fully Static, Single-Chip CMOS Microcontroller	2-5
COP311L	Single-Chip N-Channel Microcontroller	2-5
COP320	Single-Chip N-Channel Microcontroller	2-23
COP320C	Single-Chip CMOS Microcontroller	2-45
COP320L	Single-Chip N-Channel Microcontroller	2-64
COP321	Single-Chip N-Channel Microcontroller	2-23
COP321C	Single-Chip CMOS Microcontroller	2-45
COP321L	Single-Chip N-Channel Microcontroller	2-64
COP322	Single-Chip N-Channel Microcontroller	2-23
COP322L	Single-Chip N-Channel Microcontroller	2-64
COP340	Single-Chip N-Channel Microcontroller	2-88
COP341	Single-Chip N-Channel Microcontroller	2-88
COP342	Single-Chip N-Channel Microcontroller	2-88
COP344C	Single-Chip CMOS Microcontroller	2-111
COP344L	Single-Chip N-Channel Microcontroller	2-112
COP345C	Single-Chip CMOS Microcontroller	2-111
COP345C	Single-Chip N-Channel Microcontroller	2-112
COP352	Frequency Generator and Counter	5-13
COP353	Frequency Generator and Counter	5-13
COP370	V.F. Display Driver	5-44
COP398	Low Power CMOS RAM and Timer (RAT™)	5-58
COP399	Low Power CMOS Memory	5-58
COP401L	ROMless N-Channel Microcontroller	3-3
COP402	ROMless N-Channel Microcontroller	3-17
COP402M	ROMless N-Channel Microcontroller	3-17
COP404	ROMless N-Channel Microcontroller	3-36
COP404L	ROMless N-Channel Microcontroller	3-46
COP410C	Fully Static, Single-Chip CMOS Microcontroller	2-3
COP410L	Single-Chip N-Channel Microcontroller	2-5
COP411C	Fully Static, Single-Chip CMOS Microcontroller	2-3
COP411L	Single-Chip N-Channel Microcontroller	2-5
COP420	Single-Chip N-Channel Microcontroller	2-23
COP420C	Single-Chip CMOS Microcontroller	2-45
COP420L	Single-Chip N-Channel Microcontroller	2-64



## Alphanumerical Index (continued)

COP420R	Piggyback-EPROM	4-3
COP421	Single-Chip N-Channel Microcontroller	2-23
COP421C	Single-Chip CMOS Microcontroller	2-45
COP421L	Single-Chip N-Channel Microcontroller	2-64
COP422	Single-Chip N-Channel Microcontroller	2-23
COP422L	Single-Chip N-Channel Microcontroller	2-64
COP431	8-Bit Serial I/O A/D Converter	5-3
COP432	8-Bit Serial I/O A/D Converter with Multiplexer Options	5-3
COP434	8-Bit Serial I/O A/D Converter with Multiplexer Options	5-3
COP438	8-Bit Serial I/O A/D Converter with Multiplexer Options	5-3
COP440	Single-Chip N-Channel Microcontroller	2-88
COP440R	Piggyback-EPROM Microcontroller	4-18
COP441	Single-Chip N-Channel Microcontroller	2-88
COP442	Single-Chip N-Channel Microcontroller	2-88
COP444C	Single-Chip CMOS Microcontroller	2-111
COP444L	Single-Chip N-Channel Microcontroller	2-112
COP444LR	Piggyback-EPROM Microcontroller	4-3
COP445C	Single-Chip CMOS Microcontroller	2-111
COP445L	Single-Chip N-Channel Microcontroller	2-112
COP452	Frequency Generator and Counter	5-13
COP453	Frequency Generator and Counter	5-13
COP464	Single-Chip 3k Microcontroller	2-134
COP470	V.F. Display Driver	5-44
COP472	Liquid Crystal Display Controller	5-51
COP484	Single-Chip 4k Microcontroller	2-134
COP498	Low Power CMOS RAM and Timer (RAT™)	5-58
COP499	Low Power CMOS Memory	5-58
COP2304	ROMless Dual CPU Microcontroller	3-65
COP2340	Single-Chip Dual CPU Microcontroller	2-135
COP2341	Single-Chip Dual CPU Microcontroller	2-135
COP2342	Single-Chip Dual CPU Microcontroller	2-135
COP2404	ROMless Dual CPU Microcontroller	3-65
COP2440	Single-Chip Dual CPU Microcontroller	2-135
COP2440R	Piggyback-EPROM Microcontroller	4-18
COP2441	Single-Chip Dual CPU Microcontroller	2-135
COP2442	Single-Chip Dual CPU Microcontroller	2-135
DM54LS373	Octal D-Type Transparent Latch and Edge-Triggered Flip-Flop	7-27
DM54LS374	Octal D-Type Transparent Latch and Edge-Triggered Flip-Flop	7-27
DM74LS373	Octal D-Type Transparent Latch and Edge-Triggered Flip-Flop	7-27
DM74LS374	Octal D-Type Transparent Latch and Edge-Triggered Flip-Flop	7-27



## Alphanumerical Index (continued)

DS8906	AM/FM Digital Phase-Locked Loop Synthesizer . . . . .	5-70
DS8907	AM/FM Digital Phase-Locked Loop Frequency Synthesizer . . . . .	5-76
DS8908	AM/FM Digital Phase-Locked Loop Frequency Synthesizer . . . . .	5-82
MM2716	16,384-Bit (2048 × 8) UV Erasable PROM . . . . .	6-3
MM2758	8192-Bit (1024 × 8) UV Erasable PROM . . . . .	7-15
MM5445	V.F. Display Driver . . . . .	5-89
MM5446	V.F. Display Driver . . . . .	5-89
MM5447	V.F. Display Driver . . . . .	5-89
MM5448	V.F. Display Driver . . . . .	5-89
MM5450	LED Display Driver . . . . .	5-92
MM5451	LED Display Driver . . . . .	5-92
MM5452	Liquid Crystal Display Driver . . . . .	5-97
MM5453	Liquid Crystal Display Driver . . . . .	5-97
MM5480	LED Display Driver . . . . .	5-102
MM5481	LED Display Driver . . . . .	5-105
MM5484	16-Segment LED Display Driver . . . . .	5-108
MM5485	11-Segment LED Display Driver . . . . .	5-108
MM57409	Super Number Cruncher . . . . .	6-3
MM57436	Decimal/Binary Up/Down Counter . . . . .	6-6
MM57455	Advanced Educational Arithmetic Game . . . . .	6-13
MM57459	8-Digit LED Direct-Drive Memory Calculator . . . . .	6-16
MM58201	Multiplexed LCD Driver . . . . .	5-111
MM58241	High Voltage Display Driver . . . . .	5-116
MM58248	High Voltage Display Driver . . . . .	5-116
MM58341	High Voltage Display Driver . . . . .	5-119
MM58348	High Voltage Display Driver . . . . .	5-119
MM54C373	TRI-STATE® Octal D-Type Latch . . . . .	7-21
MM54C374	TRI-STATE® Octal D-Type Flip-Flop . . . . .	7-21
MM74C373	TRI-STATE® Octal D-Type Latch . . . . .	7-21
MM74C374	TRI-STATE® Octal D-Type Flip-Flop . . . . .	7-21
NMC27C16	16,384-Bit (2048 × 8) UV Erasable CMOS PROM . . . . .	7-9





**Section 1**  
**Introduction**  
**COPS Family**





## National Semiconductor COPSTM Microcontroller Family Guide

COP:	410L	410C	411L	411C	420	420L	420C	Future 420C	421	421L	421C	Future 421C	422	422L
ROM × 8	512								1024					
RAM × 4	32								64					
Inputs	0				4				0					
Bidirectional TRI-STATE <sup>®</sup> I/O	8								8					
Bidirectional I/O	4		3		4				4		2			
Outputs	4		2		4				4		2			
Serial I/O and External Event Counter	Yes				Yes		SIO		Yes		SIO		Yes	
Internal Time Base Counter	No								Yes					
Time Base Counter Programmable	No				No		Yes		No		Yes		No	
Interrupt	No				Yes				No					
Stack Levels	2								3					
MICROBUS <sup>™</sup> Option	No				Yes		No		Yes		No			
Instruction Cycle (μs) Min.—Max.	15-40	4-DC	15-40	4-DC	4-10	15-40	15-245	4-DC	4-10	15-40	15-245	4-DC	4-10	15-40
Package Size (pins)	24		20		28				24				20	
Availability	Now	Future	Now	Future	Now		Future		Now		Future		Now	

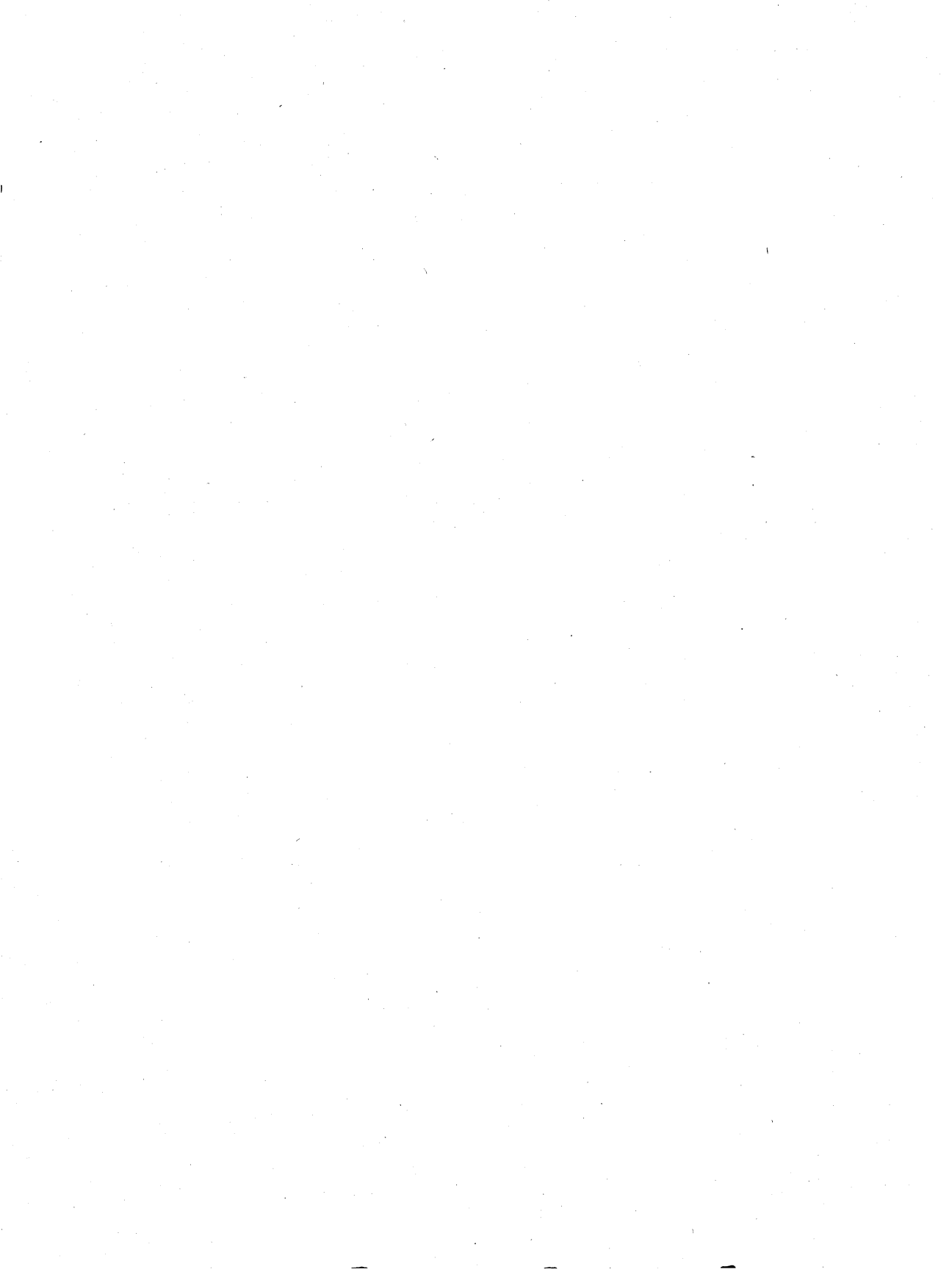
## National Semiconductor COPSTM Microcontroller Family Guide (continued)

COP:	444L	444C	445L	445C	440	441	442	2440	2441	2442	464	465	484	485		
ROM × 8					2048								3072		4096	
RAM × 4	128								160				192		256	
Inputs	4		0		4		0		4		0		4		0	
Bidirectional TRI-STATE <sup>®</sup> I/O	8				16		8		16		8		8		8	
Bidirectional I/O	4				8		4		8		4		4		4	
Outputs	4				4				4				4			
Serial I/O and External Event Counter	Yes								Yes				Yes		Yes	
Internal Time Base Counter	Yes								Yes				Yes		Yes	
Time Base Counter Programmable	No	Yes	No	Yes	Yes				No				No			
Interrupt	Yes		No		Yes 4 Sources		Yes 2 Sources		Yes 4 Sources		Yes 2 Sources		Yes		No	
Stack Levels	3				4				4 per CPU				4		4	
MICROBUS <sup>™</sup> Option	No	Yes	No		Yes		No		Yes		No		No		No	
Instruction Cycle (μs) Min.—Max.	15-40	4-DC	15-40	4-DC	4-10				4-25				4-25			
Package Size (pins)	28		24		40	28	24	40	28	24	28	24	28	24		
Availability	Now	Future	Now	Future	Now				Future				Future			

## National Semiconductor COPSTM ROMless Microcontroller Family Guide

COP:	401L	402	402M	404C	404L	404	2404	408	409
External ROM X8	Up to 512	Up to 1024		Up to 2048				Up to 4096	Up to 32768
RAM × 4	32	64		128		160		256	512
Inputs	0	4		4		4		4	
Bidirectional TRI-STATE <sup>®</sup> I/O	8	8		8		16		8	
Bidirectional I/O	4	4		4		8		4	
Outputs	4	4		4		4		4	
Serial I/O and External Event Counter	Yes	Yes		Yes		Yes		Yes	
Internal Time Base Counter	No	Yes		Yes		Yes		Yes	
Time Base Counter Programmable	No	No		Yes	No		Yes	No	
Interrupt	No	Yes	No	Yes		Yes—4 sources		Yes	
Stack Levels	2	3		3		4	4 per CPU		8
MICROBUS <sup>™</sup> Option	No	No	Yes	Yes	No		Yes	No	
Instruction Cycle (μs) Min.—Max.	15-40	4-10		4-DC	15-40		4-10	4-25	
Package Size (pins)	40	40		40/48	40		48	40	
Availability	Now	Now		Future	Now		Now	Future	

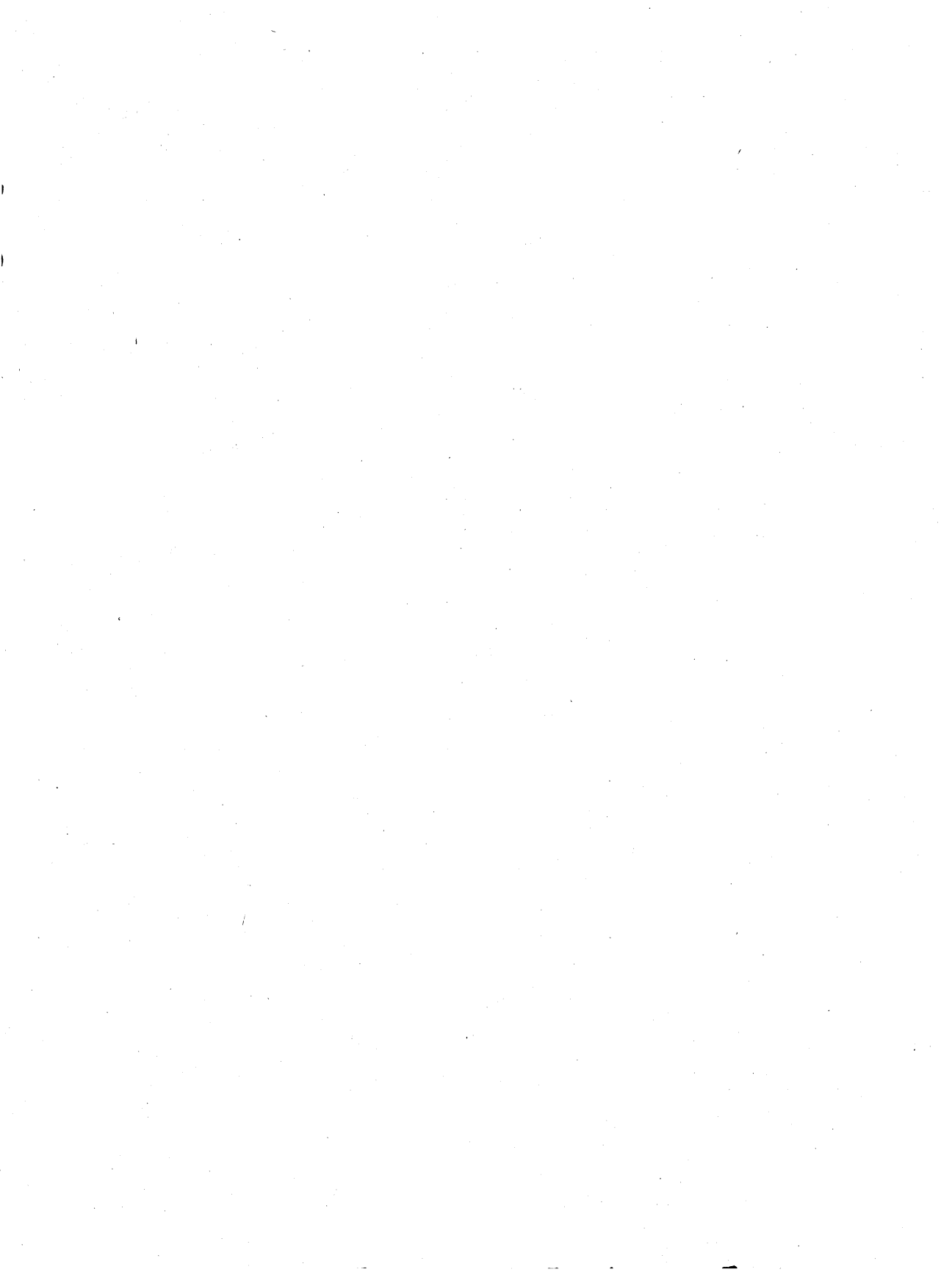






Section 2  
**Single-Chip  
Microcontrollers**





# COP410C/COP411C and COP310C/COP311C Fully Static, Single-Chip CMOS Microcontrollers

## General Description

The COP410C, COP411C, COP310C, and COP311C fully static, single-chip CMOS microcontrollers are members of the COPSTM family, fabricated using double-poly, silicon gate complementary MOS technology. These microcontrollers are complete microcomputers containing all system timing, internal logic, ROM, RAM, and I/O necessary to implement dedicated control functions in a variety of output configuration options, with an instruction set, internal architecture, and I/O scheme designed to facilitate keyboard input, display output, and BCD data manipulation. The COP411C is identical to the COP410C but with 16 I/O lines instead of 19. They are an appropriate choice for use in numerous human interface control environments. Standard test procedures and reliable high-density fabrication techniques provide the medium to large volume customers with a customized Controller Oriented Processor at a low end product cost.

The COP310C/COP311C are exact functional equivalents, but extended temperature range versions of the COP410C/COP411C.

TRI-STATE is a registered trademark of National Semiconductor Corp.  
COPS and MICROWIRE are trademarks of National Semiconductor Corp.

## Features

- Lowest power dissipation (40  $\mu$ W typical)
- Low cost
- Power saving HALT mode with Continue function
- Powerful instruction set
- 512  $\times$  8 ROM, 32  $\times$  4 RAM
- 19 I/O lines (COP410C)
- Two-level subroutine stack
- DC to 4  $\mu$ s instruction time
- Single supply operation (2.4V to 5.5V)
- General purpose and TRI-STATE<sup>®</sup> outputs
- Internal binary counter register with MICROWIRE<sup>™</sup> serial I/O capability
- LSTTL/CMOS compatible in and out
- Software/hardware compatible with other members of the COP400 family
- MICROWIRE<sup>™</sup> compatible serial I/O
- Extended temperature range device available (-40°C to +85°C)

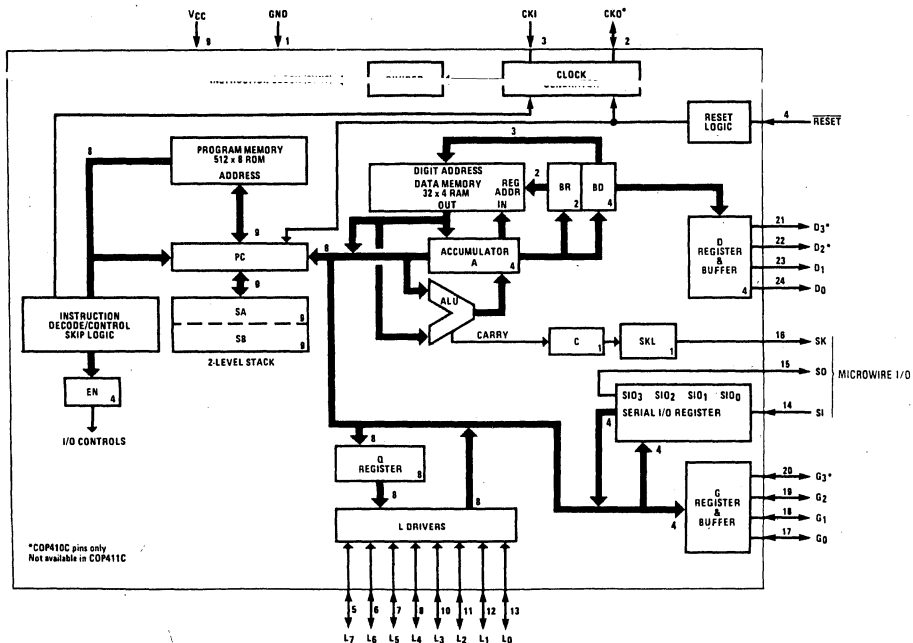
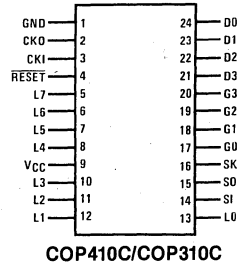
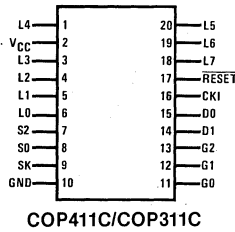


Figure 1. COP410C/COP411C Block Diagram

## Connection Diagrams



## Functional Description

### Oscillator

There are three basic clock oscillator configurations:

- Crystal Controlled Oscillator
- External Oscillator
- RC Controlled Oscillator

### HALT Mode

The COP410C/COP411C is a *fully static* circuit; therefore, the user is able to either stop the system oscillator input (CKI), or place the device in its "HALT" mode by either software or hardware control. Once in the HALT mode, the internal circuitry does not receive any clock signal, and is therefore frozen in the exact state it was in at the moment of the HALT stimulus. Since the circuit is fully static, all information is retained. The HALT mode is also the minimum power dissipation state of the device.

### I/O Options

- Standard (Push-Pull)** — An N-channel device to ground in conjunction with a P-channel device to  $V_{CC}$ , compatible with CMOS and LSTTL.
- Low Current** — This is the same as **a)** above except that the source current is approximately ten times smaller.
- Open Drain** — An N-channel device to ground only, allowing external pull-up as required by the user's application.

- Standard TRI-STATE® L Output** — A CMOS output buffer which may be disabled by program control.
- Low Current TRI-STATE L Output** — This is the same as **d)** above except that the source current is approximately ten times smaller.
- Open Drain TRI-STATE L Output** — This has only the N-channel device to ground, which may be disabled by program control.
- An on-chip pull-up load device to  $V_{CC}$  (input option).
- A Hi-Z input which must be driven by user logic.

### CKO Pin Options

In a crystal-controlled oscillator system, CKO is used as an output to the crystal network. CKO will be forced high during the execution of a HALT instruction, thus inhibiting the crystal network. If a one-pin oscillator system is chosen (RC or external), CKO will be a conversational I/O port used to flag the execution of a HALT instruction. CKO can at any time and in any clock configuration be externally forced high to execute a Hardware Halt, but the continue function (force CKO low to restart the device) is only available when using a one-pin oscillator.

## Instruction Set

Exactly the same as the COP410L/COP411L with the additional instruction:

HALT Halt System Oscillator

# COP410L/COP411L and COP310L/COP311L Single-Chip N-Channel Microcontrollers

## General Description

The COP410L and COP411L Single-Chip N-Channel Microcontrollers are members of the COPS™ family, fabricated using N-channel, silicon gate MOS technology. These Controller Oriented Processors are complete microcomputers containing all system timing, internal logic, ROM, RAM and I/O necessary to implement dedicated control functions in a variety of applications. Features include single supply operation, a variety of output configuration options, with an instruction set, internal architecture and I/O scheme designed to facilitate keyboard input, display output and BCD data manipulation. The COP411L is identical to the COP410L, but with 16 I/O lines instead of 19. They are an appropriate choice for use in numerous human interface control environments. Standard test procedures and reliable high-density fabrication techniques provide the medium to large volume customers with a customized Controller Oriented Processor at a low end-product cost.

The COP310L and COP311L are exact functional equivalents but extended temperature versions of COP410L and COP411L respectively.

The COP401L may be used for exact emulation.

## Features

- Low cost
- Powerful instruction set
- 512 × 8 ROM, 32 × 4 RAM
- 19 I/O lines (COP410L)
- Two-level subroutine stack
- 16μs instruction time
- Single supply operation (4.5–6.3V)
- Low current drain (6mA max.)
- Internal binary counter register with MICROWIRE™ serial I/O capability
- General purpose and TRI-STATE® outputs
- LSTTL/CMOS compatible in and out
- Direct drive of LED digit and segment lines
- Software/hardware compatible with other members of COP400 family
- Extended temperature range device COP310L/COP311L (–40°C to +85°C)
- Wider supply range (4.5–9.5V) optionally available

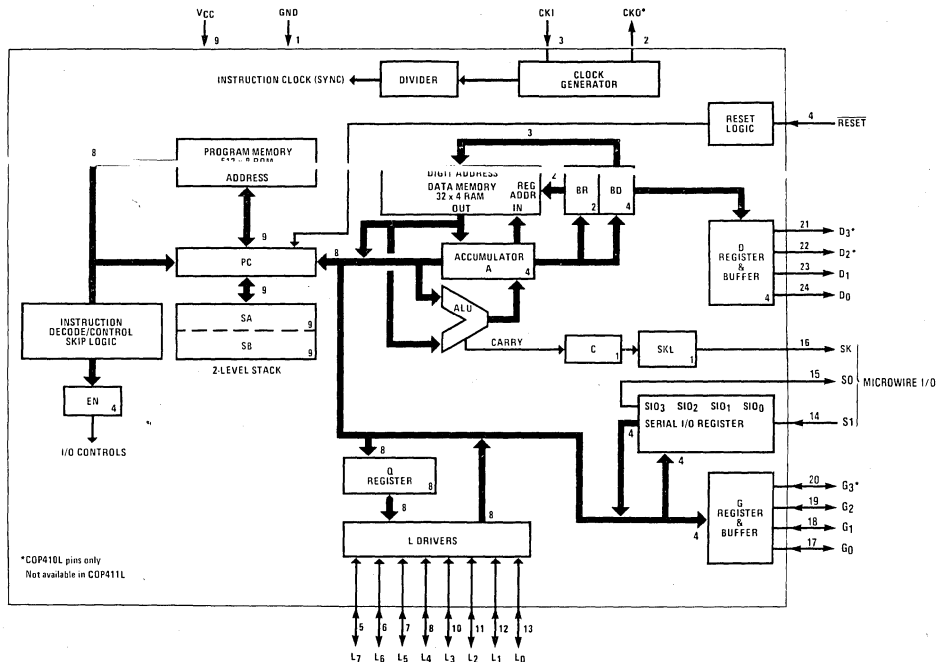


Figure 1. COP410L/411L Block Diagram

**COP410L/COP411L****Absolute Maximum Ratings**

Voltage at Any Pin Relative to GND	-0.5V to +10V
Ambient Operating Temperature	0°C to +70°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C
Power Dissipation	
COP410L	0.75 Watt at 25°C 0.4 Watt at 70°C
COP411L	0.65 Watt at 25°C 0.3 Watt at 70°C
Total Source Current	120 mA
Total Sink Current	100 mA

*Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

**DC Electrical Characteristics**  $0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 9.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Standard Operating Voltage ( $V_{CC}$ )	Note 1	4.5	6.3	V
Optional Operating Voltage ( $V_{CC}$ )		4.5	9.5	V
Power Supply Ripple	peak to peak		0.5	V
Operating Supply Current	all inputs and outputs open		6	mA
Input Voltage Levels				
CKI Input Levels				
Ceramic Resonator Input ( $\div 8$ )				
Logic High ( $V_{IH}$ )		2.0		V
Logic Low ( $V_{IL}$ )		-0.3	0.4	V
Schmitt Trigger Input ( $\div 4$ )				
Logic High ( $V_{IH}$ )		0.7 $V_{CC}$		V
Logic Low ( $V_{IL}$ )		-0.3	0.6	V
RESET Input Levels	(Schmitt Trigger Input)			
Logic High		0.7 $V_{CC}$		V
Logic Low		-0.3	0.6	V
SO Input Level (Test mode)	Note 2	2.0	2.5	V
All Other Inputs				
Logic High	$V_{CC} = \text{Max.}$	3.0		V
Logic High	with TTL trip level options	2.0		V
Logic Low	selected, $V_{CC} = 5\text{V} \pm 5\%$	-0.3	0.8	V
Logic High	with high trip level options	3.6		V
Logic Low	selected	-0.3	1.2	V
Input Capacitance			7	pF
Hi-Z Input Leakage		-1	+1	$\mu\text{A}$
Output Voltage Levels				
LSTTL Operation	$V_{CC} = 5\text{V} \pm 5\%$			
Logic High ( $V_{OH}$ )	$I_{OH} = -25\mu\text{A}$	2.7		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 0.36\text{mA}$		0.4	V
CMOS Operation				
Logic High	$I_{OH} = -10\mu\text{A}$	$V_{CC} - 1$		V
Logic Low	$I_{OL} = +10\mu\text{A}$		0.2	V

**Note 1:**  $V_{CC}$  voltage change must be less than 0.5V in a 1ms period to maintain proper operation.

**Note 2:** SO output "0" level must be less than 0.8V for normal operation.

## COP410L/COP411L

DC Electrical Characteristics (continued)  $0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 9.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
<b>Output Current Levels</b>				
<b>Output Sink Current</b>				
SO and SK Outputs ( $I_{OL}$ )	$V_{CC} = 9.5\text{V}$ , $V_{OL} = 0.4\text{V}$	1.8		mA
	$V_{CC} = 6.3\text{V}$ , $V_{OL} = 0.4\text{V}$	1.2		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.9		mA
$L_0$ - $L_7$ Outputs, $G_0$ - $G_3$ and LSTTL $D_0$ - $D_3$ Outputs ( $I_{OL}$ )	$V_{CC} = 9.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.8		mA
	$V_{CC} = 6.3\text{V}$ , $V_{OL} = 0.4\text{V}$	0.5		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.4		mA
$D_0$ - $D_3$ Outputs with High Current Options ( $I_{OL}$ )	$V_{CC} = 9.5\text{V}$ , $V_{OL} = 1.0\text{V}$	15		mA
	$V_{CC} = 6.3\text{V}$ , $V_{OL} = 1.0\text{V}$	11		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 1.0\text{V}$	7.5		mA
$D_0$ - $D_3$ Outputs with Very High Current Options ( $I_{OL}$ )	$V_{CC} = 9.5\text{V}$ , $V_{OL} = 1.0\text{V}$	30		mA
	$V_{CC} = 6.3\text{V}$ , $V_{OL} = 1.0\text{V}$	22		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 1.0\text{V}$	15		mA
CKI (Single-pin RC oscillator)	$V_{CC} = 4.5\text{V}$ , $V_{IH} = 3.5\text{V}$	2		mA
CKO	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.2		mA
<b>Output Source Current</b>				
<b>Standard Configuration, All Outputs (<math>I_{OH}</math>)</b>				
	$V_{CC} = 9.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-140	-800	$\mu\text{A}$
	$V_{CC} = 6.3\text{V}$ , $V_{OH} = 2.0\text{V}$	-75	-480	$\mu\text{A}$
	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-30	-250	$\mu\text{A}$
<b>Push-Pull Configuration, SO and SK Outputs (<math>I_{OH}</math>)</b>				
	$V_{CC} = 9.5\text{V}$ , $V_{OH} = 4.75\text{V}$	-1.4		mA
	$V_{CC} = 6.3\text{V}$ , $V_{OH} = 2.4\text{V}$	-1.4		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 1.0\text{V}$	-1.2		mA
<b>LED Configuration, <math>L_0</math>-<math>L_7</math> Outputs, Low Current Driver Option (<math>I_{OH}</math>)</b>				
	$V_{CC} = 9.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-1.5	-18	mA
	$V_{CC} = 6.0\text{V}$ , $V_{OH} = 2.0\text{V}$	-1.5	-13	mA
<b>LED Configuration, <math>L_0</math>-<math>L_7</math> Outputs, High Current Driver Option (<math>I_{OH}</math>)</b>				
	$V_{CC} = 9.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-3.0	-35	mA
	$V_{CC} = 6.0\text{V}$ , $V_{OH} = 2.0\text{V}$	-3.0	-25	mA
<b>TRI-STATE<sup>®</sup> Configuration, <math>L_0</math>-<math>L_7</math> Outputs, Low Current Driver Option (<math>I_{OH}</math>)</b>				
	$V_{CC} = 9.5\text{V}$ , $V_{OH} = 5.5\text{V}$	-0.75		mA
	$V_{CC} = 6.3\text{V}$ , $V_{OH} = 3.2\text{V}$	-0.8		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 1.5\text{V}$	-0.9		mA
<b>TRI-STATE<sup>®</sup> Configuration, <math>L_0</math>-<math>L_7</math> Outputs, High Current Driver Option (<math>I_{OH}</math>)</b>				
	$V_{CC} = 9.5\text{V}$ , $V_{OH} = 5.5\text{V}$	-1.5		mA
	$V_{CC} = 6.3\text{V}$ , $V_{OH} = 3.2\text{V}$	-1.6		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 1.5\text{V}$	-1.8		mA
Input Load Source Current	$V_{CC} = 5.0\text{V}$ , $V_{IL} = 0\text{V}$	-10	-140	$\mu\text{A}$
<b>CKO Output</b>				
<b>RAM Power Supply Option Power Requirement</b>				
	$V_R = 3.3\text{V}$		1.5	mA
TRI-STATE <sup>®</sup> Output Leakage Current		-2.5	+2.5	$\mu\text{A}$
<b>Total Sink Current Allowed</b>				
<b>All Outputs Combined</b>				
D Port			100	mA
$L_7$ - $L_4$ , G Port			100	mA
$L_3$ - $L_0$			4	mA
Any Other Pin			4	mA
<b>Total Source Current Allowed</b>				
<b>All I/O Combined</b>				
$L_7$ - $L_4$			120	mA
$L_3$ - $L_0$			60	mA
Each L Pin			60	mA
Any Other Pin			25	mA
			1.5	mA



## COP310L/COP311L

## Absolute Maximum Ratings

Voltage at Any Pin Relative to GND	-0.5V to +10V
Ambient Operating Temperature	-40°C to +85°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C
Power Dissipation	
COP310L	0.75 Watt at 25°C 0.25 Watt at 85°C
COP311L	0.65 Watt at 25°C 0.20 Watt at 85°C
Total Source Current	120 mA
Total Sink Current	100 mA

*Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

DC Electrical Characteristics -40°C ≤ T<sub>A</sub> ≤ +85°C, 4.5V ≤ V<sub>CC</sub> ≤ 7.5V unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Standard Operating Voltage (V <sub>CC</sub> )	Note 1	4.5	5.5	V
Optional Operating Voltage (V <sub>CC</sub> )		4.5	7.5	V
Power Supply Ripple	peak to peak		0.5	V
Operating Supply Current	all inputs and outputs open		8	mA
Input Voltage Levels				
Ceramic Resonator Input (+8)				
Crystal Input				
Logic High (V <sub>IH</sub> )		2.2		V
Logic Low (V <sub>IL</sub> )		-0.3	0.3	V
Schmitt Trigger Input (+4)				
Logic High (V <sub>IH</sub> )		0.7 V <sub>CC</sub>		V
Logic Low (V <sub>IL</sub> )		-0.3	0.4	V
RESET Input Levels	(Schmitt Trigger Input)			
Logic High		0.7 V <sub>CC</sub>		V
Logic Low		-0.3	0.4	V
SO Input Level (Test mode)	Note 2	2.2	2.5	V
All Other Inputs				
Logic High	V <sub>CC</sub> = Max.	3.0		V
Logic High	with TTL trip level options	2.2		V
Logic Low	selected, V <sub>CC</sub> = 5V ± 5%	-0.3	0.6	V
Logic High	with high trip level options	3.6		V
Logic Low	selected	-0.3	1.2	V
Input Capacitance			7	pF
Hi-Z Input Leakage		-2	+2	μA
Output Voltage Levels				
LSTTL Operation	V <sub>CC</sub> = 5V ± 5%			
Logic High (V <sub>OH</sub> )	I <sub>OH</sub> = -20 μA	2.7		V
Logic Low (V <sub>OL</sub> )	I <sub>OL</sub> = 0.36 mA		0.4	V
CMOS Operation				
Logic High	I <sub>OH</sub> = -10 μA	V <sub>CC</sub> - 1		V
Logic Low	I <sub>OL</sub> = +10 μA		0.2	V

**Note 1:** V<sub>CC</sub> voltage change must be less than 0.5V in a 1ms period to maintain proper operation.

**Note 2:** SO output "0" level must be less than 0.6V for normal operation.

# COP310L/COP311L

## DC Electrical Characteristics (continued) $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ , $4.5\text{V} \leq V_{CC} \leq 7.5\text{V}$ unless otherwise noted.

COP410L/COP411L, COP310L/COP311L

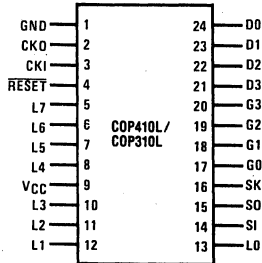
2

Parameter	Conditions	Min.	Max.	Units
<b>Output Current Levels</b>				
<b>Output Sink Current</b>				
SO and SK Outputs ( $I_{OL}$ )	$V_{CC} = 7.5\text{V}$ , $V_{OL} = 0.4\text{V}$	1.4		mA
	$V_{CC} = 5.5\text{V}$ , $V_{OL} = 0.4\text{V}$	1.0		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.8		mA
$L_0$ - $L_7$ Outputs, $G_0$ - $G_3$ and LSTTL, $D_0$ - $D_3$ Outputs ( $I_{OL}$ )	$V_{CC} = 7.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.6		mA
	$V_{CC} = 5.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.5		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.4		mA
$D_0$ - $D_3$ Outputs with High Current Options ( $I_{OL}$ )	$V_{CC} = 7.5\text{V}$ , $V_{OL} = 1.0\text{V}$	12		mA
	$V_{CC} = 5.5\text{V}$ , $V_{OL} = 1.0\text{V}$	9		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 1.0\text{V}$	7		mA
$D_0$ - $D_3$ Outputs with Very High Current Options ( $I_{OL}$ )	$V_{CC} = 7.5\text{V}$ , $V_{OL} = 1.0\text{V}$	24		mA
	$V_{CC} = 5.5\text{V}$ , $V_{OL} = 1.0\text{V}$	18		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 1.0\text{V}$	14		mA
CKI (Single-pin RC oscillator)	$V_{CC} = 4.5\text{V}$ , $V_{IH} = 3.5\text{V}$	1.5		mA
CKO	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.2		mA
<b>Output Source Current</b>				
Standard Configuration, All Outputs ( $I_{OH}$ )	$V_{CC} = 7.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-100	-900	$\mu\text{A}$
	$V_{CC} = 5.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-55	-600	$\mu\text{A}$
	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-28	-350	$\mu\text{A}$
Push-Pull Configuration SO and SK Outputs ( $I_{OH}$ )	$V_{CC} = 7.5\text{V}$ , $V_{OH} = 3.75\text{V}$	-0.85		mA
	$V_{CC} = 5.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-1.1		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 1.0\text{V}$	-1.2		mA
LED Configuration, $L_0$ - $L_7$ Outputs, Low Current Driver Option ( $I_{OH}$ )	$V_{CC} = 7.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-1.4	-27	mA
	$V_{CC} = 5.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-0.7	-15	$\mu\text{A}$
LED Configuration, $L_0$ - $L_7$ Outputs, High Current Driver Option ( $I_{OH}$ )	$V_{CC} = 7.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-2.7	-54	mA
	$V_{CC} = 5.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-1.4	-30	$\mu\text{A}$
TRI-STATE® Configuration, $L_0$ - $L_7$ Outputs, Low Current Driver Option ( $I_{OH}$ )	$V_{CC} = 7.5\text{V}$ , $V_{OH} = 4.0\text{V}$	-0.7		mA
	$V_{CC} = 5.5\text{V}$ , $V_{OH} = 2.7\text{V}$	-0.6		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 1.5\text{V}$	-0.9		mA
TRI-STATE® Configuration, $L_0$ - $L_7$ Outputs, High Current Driver Option ( $I_{OH}$ )	$V_{CC} = 7.5\text{V}$ , $V_{OH} = 4.0\text{V}$	-1.4		mA
	$V_{CC} = 5.5\text{V}$ , $V_{OH} = 2.7\text{V}$	-1.2		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 1.5\text{V}$	-1.8		mA
Input Load Source Current	$V_{CC} = 5.0\text{V}$ , $V_{IL} = 0\text{V}$	-10	-200	$\mu\text{A}$
CKO Output				
RAM Power Supply Option Power Requirement	$V_R = 3.3\text{V}$		2.0	mA
TRI-STATE® Output Leakage Current		-5	+5	$\mu\text{A}$
<b>Total Sink Current Allowed</b>				
All Outputs Combined			100	mA
D Port			100	mA
$L_7$ - $L_4$ , G Port			4	mA
$L_3$ - $L_0$			4	mA
All Other Pins			1.5	mA
<b>Total Source Current Allowed</b>				
All I/O Combined			120	mA
$L_7$ - $L_4$			60	mA
$L_3$ - $L_0$			60	mA
Each L Pin			25	mA
All Other Pins			1.5	mA

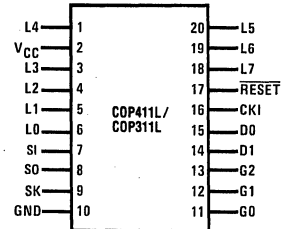
## AC Electrical Characteristics

COP410L/411L:  $0^{\circ}\text{C} \leq T_A \leq 70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 9.5\text{V}$  unless otherwise noted.  
 COP310L/311L:  $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 7.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Instruction Cycle Time — $t_C$		15	40	$\mu\text{s}$
CKI				
Input Frequency — $f_i$	$\div 8$ mode	0.2	0.5	MHz
	$\div 4$ mode	0.1	0.26	MHz
Duty Cycle		30	60	%
Rise Time	$f_i = 0.5\text{MHz}$		500	ns
Fall Time			200	ns
CKI Using RC ( $\div 4$ )	$R = 56\text{k}\Omega \pm 5\%$ $C = 100\text{pF} \pm 10\%$			
Instruction Cycle Time		15	28	$\mu\text{s}$
CKO as SYNC Input				
$t_{\text{SYNC}}$		400		ns
INPUTS:				
$G_3-G_0, L_7-L_0$			8.0	$\mu\text{s}$
$t_{\text{SETUP}}$			1.3	$\mu\text{s}$
$t_{\text{HOLD}}$				
SI			2.0	$\mu\text{s}$
$t_{\text{SETUP}}$			1.0	$\mu\text{s}$
$t_{\text{HOLD}}$				
OUTPUT PROPAGATION DELAY	Test condition: $C_L = 50\text{pF}, R_L = 20\text{k}\Omega, V_{\text{OUT}} = 1.5\text{V}$			
SO, SK Outputs			4.0	$\mu\text{s}$
$t_{\text{pd1}}, t_{\text{pd0}}$				
All Other Outputs			5.6	$\mu\text{s}$
$t_{\text{pd1}}, t_{\text{pd0}}$				



Order Number COP410L/N, COP310L/N  
NS Package N24A



Order Number COP411L/N, COP311L/N  
NS Package N20A

Pin	Description	Pin	Description
$L_7-L_0$	8 bidirectional I/O ports with TRI-STATE <sup>®</sup>	SK	Logic-controlled clock (or general purpose output)
$G_3-G_0$	4 bidirectional I/O ports ( $G_2-G_0$ for COP411L)	CKI	System oscillator input
$D_3-D_0$	4 general purpose outputs ( $D_1-D_0$ for COP411L)	CKO	System oscillator output (or RAM power supply or SYNC input) (COP410L only)
SI	Serial input (or counter input)	RESET	System reset input
SO	Serial output (or general purpose output)	VCC	Power supply
		GND	Ground

Figure 2. Connection Diagrams

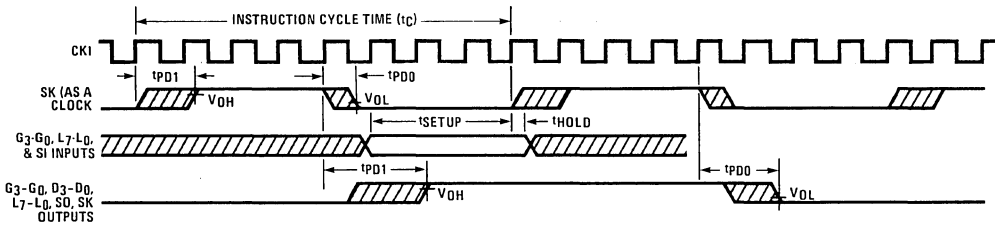


Figure 3. Input/Output Timing Diagrams (Ceramic Resonator Divide-by-8 Mode)

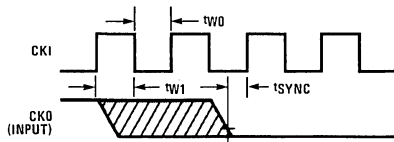


Figure 3a. Synchronization Timing

## Functional Description

A block diagram of the COP410L is given in Figure 1. Data paths are illustrated in simplified form to depict how the various logic elements communicate with each other in implementing the instruction set of the device. Positive logic is used. When a bit is set, it is a logic "1" (greater than 2 volts). When a bit is reset, it is a logic "0" (less than 0.8 volts).

All functional references to the COP410L/COP411L also apply to the COP310L/COP311L.

### Program Memory

Program Memory consists of a 512-byte ROM. As can be seen by an examination of the COP410L/411L instruction set, these words may be program instructions, program data or ROM addressing data. Because of the special characteristics associated with the JP, JSRP, JID and LQID instructions, ROM must often be thought of as being organized into 8 pages of 64 words each.

ROM addressing is accomplished by a 9-bit PC register. Its binary value selects one of the 512 8-bit words contained in ROM. A new address is loaded into the PC register during each instruction cycle. Unless the instruction is a transfer of control instruction, the PC register is loaded with the next sequential 9-bit binary count value. Two levels of subroutine nesting are implemented by the 9-bit subroutine save registers, SA and SB, providing a last-in, first-out (LIFO) hardware subroutine stack.

ROM instruction words are fetched, decoded and executed by the Instruction Decode, Control and Skip Logic circuitry.

### Data Memory

Data memory consists of a 128-bit RAM, organized as 4 data registers of 8 4-bit digits. RAM addressing is implemented by a 6-bit B register whose upper 2 bits (Br

select 1 of 4 data registers and lower 3 bits of the 4-bit Bd select 1 of 8 4-bit digits in the selected data register. While the 4-bit contents of the selected RAM digit (M) is usually loaded into or from, or exchanged with, the A register (accumulator), it may also be loaded into the Q latches or loaded from the L ports. RAM addressing may also be performed directly by the XAD 3,15 instruction. The Bd register also serves as a source register for 4-bit data sent directly to the D outputs.

The most significant bit of Bd is not used to select a RAM digit. Hence each physical digit of RAM may be selected by two different values of Bd as shown in Figure 4 below. The skip condition for XIS and XDS instructions will be true if Bd changes between 0 and 15, but NOT between 7 and 8 (see Table 3).

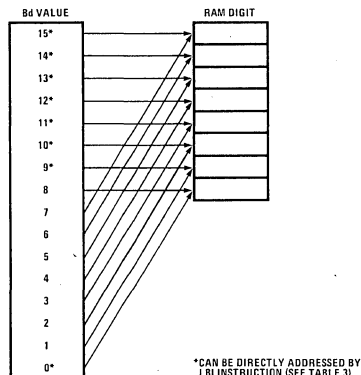


Figure 4. RAM Digit Address to Physical RAM Digit Mapping

### Internal Logic

The 4-bit A register (accumulator) is the source and destination register for most I/O, arithmetic, logic and data memory access operations. It can also be used to load the Bd portion of the B register, to load 4 bits of the 8-bit Q latch data, to input 4 bits of the 8-bit L I/O port data and to perform data exchanges with the SIO register.

A 4-bit adder performs the arithmetic and logic functions of the COP410L/411L, storing its results in A. It also outputs a carry bit to the 1-bit C register, most often employed to indicate arithmetic overflow. The C register, in conjunction with the XAS instruction and the EN register, also serves to control the SK output. C can be outputted directly to SK or can enable SK to be a sync clock each instruction cycle time. (See XAS instruction and EN register description, below.)

The G register contents are outputs to 4 general-purpose bidirectional I/O ports.

The Q register is an internal, latched, 8-bit register, used to hold data loaded from M and A, as well as 8-bit data from ROM. Its contents are output to the L I/O ports when the L drivers are enabled under program control. (See LEI instruction.)

The 8 L drivers, when enabled, output the contents of latched Q data to the L I/O ports. Also, the contents of L may be read directly into A and M. L I/O ports can be directly connected to the segments of a multiplexed LED display (using the LED Direct Drive output configuration option) with Q data being outputted to the Sa-Sg and decimal point segments of the display.

The SIO register functions as a 4-bit serial-in/serial-out shift register or as a binary counter depending on the contents of the EN register. (See EN register description, below.) Its contents can be exchanged with A, allowing it to input or output a continuous serial data stream. SIO may also be used to provide additional parallel I/O by connecting SO to external serial-in/parallel-out shift registers.

The XAS instruction copies C into the SKL Latch. In the counter mode, SK is the output of SKL in the shift register mode, SK outputs SKL ANDed with internal instruction cycle clock.

The EN register is an internal 4-bit register loaded under program control by the LEI instruction. The state of each bit of this register selects or deselects the particular feature associated with each bit of the EN register (EN<sub>3</sub>-EN<sub>0</sub>).

1. The least significant bit of the enable register, EN<sub>0</sub>, selects the SIO register as either a 4-bit shift register or a 4-bit binary counter. With EN<sub>0</sub> set, SIO is an asynchronous binary counter, *decrementing* its value by one upon each low-going pulse ("1" to "0") occurring on the SI input. Each pulse must be at least two instruction cycles wide. SK outputs the value of SKL. The SO output is equal to the value of EN<sub>3</sub>. With EN<sub>0</sub> reset, SIO is a serial shift register shifting left each instruction cycle time. The data present at SI goes into the least significant bit of SIO. SO can be enabled to output the most significant bit of SIO each cycle time. (See 4 below.) The SK output becomes a logic-controlled clock.
2. EN<sub>1</sub> is not used. It has no effect on COP410L/COP411L operation.
3. With EN<sub>2</sub> set, the L drivers are enabled to output the data in Q to the L I/O ports. Resetting EN<sub>2</sub> disables the L drivers, placing the L I/O ports in a high-impedance input state.
4. EN<sub>3</sub>, in conjunction with EN<sub>0</sub>, affects the SO output. With EN<sub>0</sub> set (binary counter option selected) SO will output the value loaded into EN<sub>3</sub>. With EN<sub>0</sub> reset (serial shift register option selected), setting EN<sub>3</sub> enables SO as the output of the SIO shift register, outputting serial shifted data each instruction time. Resetting EN<sub>3</sub> with the serial shift register option selected disables SO as the shift register output; data continues to be shifted through SIO and can be exchanged with A via an XAS instruction but SO remains reset to "0." Table 1 provides a summary of the modes associated with EN<sub>3</sub> and EN<sub>0</sub>.

### Initialization

The Reset Logic will initialize (clear) the device upon power-up if the power supply rise time is less than 1 ms and greater than 1 μs. If the power supply rise time is greater than 1 ms, the user must provide an external RC

Enable Register Modes — Bits EN<sub>3</sub> and EN<sub>0</sub>

EN <sub>3</sub>	EN <sub>0</sub>	SIO	SI	SO	SK
0	0	Shift Register	Input to Shift Register	0	If SKL = 1, SK = Clock If SKL = 0, SK = 0
1	0	Shift Register	Input to Shift Register	Serial Out	If SKL = 1, SK = Clock If SKL = 0, SK = 0
0	1	Binary Counter	Input to Binary Counter	0	If SKL = 1, SK = 1 If SKL = 0, SK = 0
1	1	Binary Counter	Input to Binary Counter	1	If SKL = 1, SK = 1 If SKL = 0, SK = 0

network and diode to the **RESET** pin as shown below (Figure 5). The **RESET** pin is configured as a Schmitt trigger input. If not used it should be connected to  $V_{CC}$ . Initialization will occur whenever a logic "0" is applied to the **RESET** input, provided it stays low for at least three instruction cycle times.

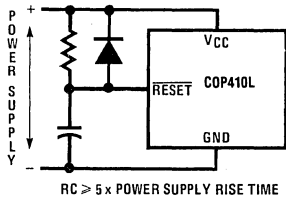


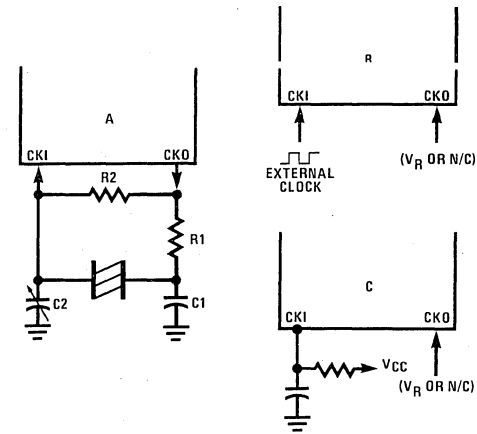
Figure 5. Power-Up Clear Circuit

Upon initialization, the PC register is cleared to 0 (ROM address 0) and the A, B, C, D, EN, and G registers are cleared. The SK output is enabled as a SYNC output, providing a pulse each instruction cycle time. *Data Memory (RAM) is not cleared upon initialization. The first instruction at address 0 must be a CLRA.*

### Oscillator

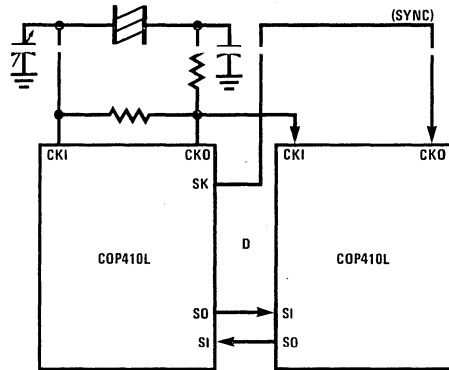
There are four basic clock oscillator configurations available as shown by Figure 6.

- a. Resonator Controlled Oscillator.** CKI and CKO are connected to an external ceramic resonator. The instruction cycle frequency equals the resonator frequency divided by 8. This is not available in the COP411L.
- b. External Oscillator.** CKI is an external clock input signal. The external frequency is divided by 8 to give the instruction frequency time. CKO is now available to be used as the RAM power supply ( $V_R$ ), as a SYNC input, or no connection. (Note: No CKO on COP411L)
- c. RC Controlled Oscillator.** CKI is configured as a single pin RC controlled Schmitt trigger oscillator. The instruction cycle equals the oscillation frequency divided by 4. CKO is available as the RAM power supply ( $V_R$ ) or no connection.
- d. Externally Synchronized Oscillator.** Intended for use in multi-COP systems, CKO is programmed to function as an input connected to the SK output of another COP chip operating at the same frequency (COP chip with L or C suffix) with CKI connected as shown. In this configuration, the SK output connected to CKO must provide a SYNC (instruction cycle) signal to CKO, thereby allowing synchronous data transfer between the COPs using only the SI and SO serial I/O pins in conjunction with the XAS instruction. Note that on power-up SK is automatically enabled as a SYNC output. (See Functional Description, Initialization, above.) This is not available in the COP411L.



Ceramic Resonator Oscillator

Resonator Value	Component Values			
	R1 ( $\Omega$ )	R2 ( $\Omega$ )	C1 (pF)	C2 (pF)
455 kHz	4.7k	1M	220	220



RC Controlled Oscillator

R (k $\Omega$ )	C (pF)	Instruction Cycle Time in $\mu$ s
51	100	19 $\pm$ 15%
82	56	19 $\pm$ 13%

Note:  $200\text{ k}\Omega \geq R \geq 25\text{ k}\Omega$   
 $360\text{ pF} \geq C \geq 50\text{ pF}$

Figure 6. COP410L/411L Oscillator

## CKO Pin Options

In a resonator controlled oscillator system, CKO is used as an output to the resonator network. As an option CKO can be a SYNC input as described above. As another option, CKO can be a RAM power supply pin ( $V_R$ ), allowing its connection to a standby/backup power supply to maintain the integrity of RAM data with minimum power drain when the main supply is inoperative or shut down to conserve power. Using no connection option is appropriate in applications where the COP410L system timing configuration does not require use of the CKO pin.

## RAM Keep-Alive Option

Selecting CKO as the RAM power supply ( $V_R$ ) allows the user to shut off the chip power supply ( $V_{CC}$ ) and maintain data in the RAM. To insure that RAM data integrity is maintained, the following conditions must be met:

1.  $\overline{\text{RESET}}$  must go low before  $V_{CC}$  goes below spec during power-off;  $V_{CC}$  must be within spec before  $\overline{\text{RESET}}$  goes high on power-up.
2. During normal operation,  $V_R$  must be within the operating range of the chip with  $(V_{CC} - 1) \leq V_R \leq V_{CC}$ .
3.  $V_R$  must be  $\geq 3.3V$  with  $V_{CC}$  off.

## I/O Options

COP410L/411L inputs and outputs have the following optional configurations, illustrated in Figure 7:

- Standard** — an enhancement-mode device to ground in conjunction with a depletion-mode device to  $V_{CC}$ , compatible with LSTTL and CMOS input requirements. Available on SO, SK, and all D and G outputs.
- Open-Drain** — an enhancement-mode device to ground only, allowing external pull-up as required by the user's application. Available on SO, SK, and all D and G outputs.
- Push-Pull** — an enhancement-mode device to ground in conjunction with a depletion-mode device paralleled by an enhancement-mode device to  $V_{CC}$ . This configuration has been provided to allow for fast rise and fall times when driving capacitive loads. Available on SO and SK outputs only.
- Standard L** — same as a., but may be disabled. Available on L outputs only.
- Open Drain L** — same as b., but may be disabled. Available on L outputs only.
- LED Direct Drive** — an enhancement mode device to ground and to  $V_{CC}$ , meeting the typical current sourcing requirements of the segments of an LED display. The sourcing device is clamped to limit current flow. These devices may be turned off under program control (see Functional Description, EN Register), placing the outputs in a high-impedance state to provide required LED segment blanking for a multiplexed display. Available on L outputs only.

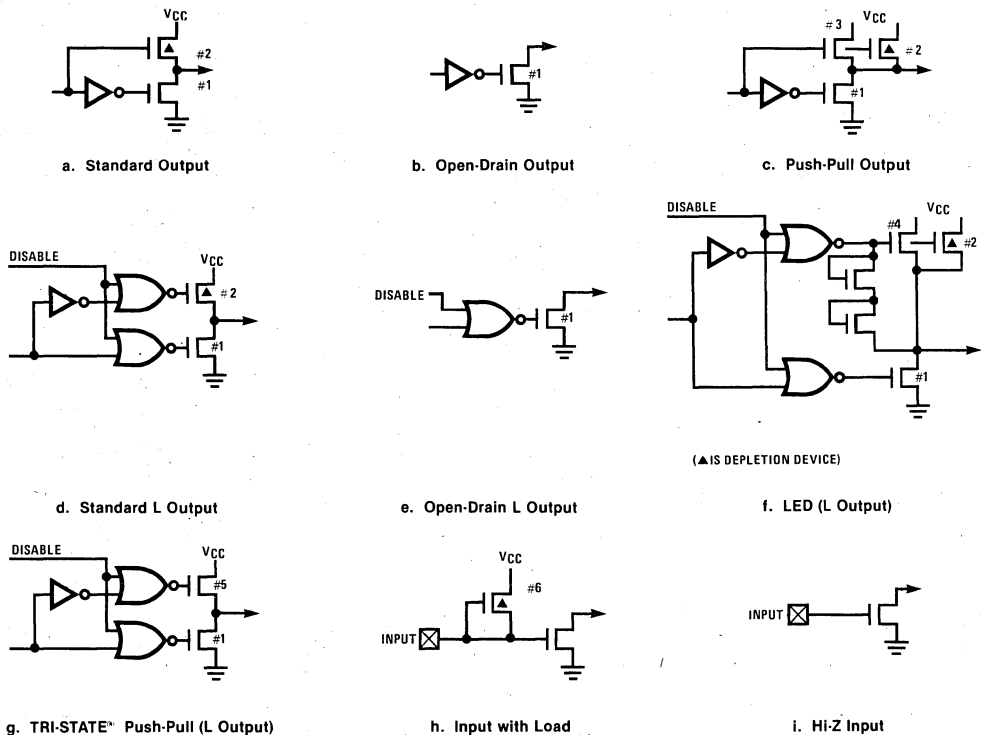


Figure 7. Input and Output Configurations

- g. TRI-STATE® Push-Pull — an enhancement-mode device to ground and  $V_{CC}$ . These outputs are TRI-STATE® outputs, allowing for connection of these outputs to a data bus shared by other bus drivers. Available on L outputs only.
- h. An on-chip depletion load device to  $V_{CC}$ .
- i. A Hi-Z input which must be driven to a "1" or "0" by external components.

The above input and output configurations share common enhancement-mode and depletion-mode devices. Specifically, all configurations use one or more of six devices (numbered 1-6, respectively). Minimum and maximum current ( $I_{OUT}$  and  $V_{OUT}$ ) curves are given in Figure 8 for each of these devices to allow the designer to effectively use these I/O configurations in designing a COP410L/411L system.

The SO, SK outputs can be configured as shown in a., b., or c. The D and G outputs can be configured as

shown in a. or b. Note that when inputting data to the G ports, the G outputs should be set to "1." The L outputs can be configured as in d., e., f., or g.

An important point to remember if using configuration d. or f. with the L drivers is that even when the L drivers are disabled, the depletion load device will source a small amount of current. (See Figure 8, device 2.) However, when the L port is used as input, the disabled depletion device CANNOT be relied on to source sufficient current to pull an input to a logic "1".

**COP411L**

If the COP410L is bonded as a 20-pin device, it becomes the COP411L, illustrated in Figure 2, COP410L/411L Connection Diagrams. Note that the COP411L does not contain D2, D3, G3, or CKO. Use of this option of course precludes use of D2, D3, G3, and CKO options. All other options are available for the COP411L.

**Typical Performance Curves**

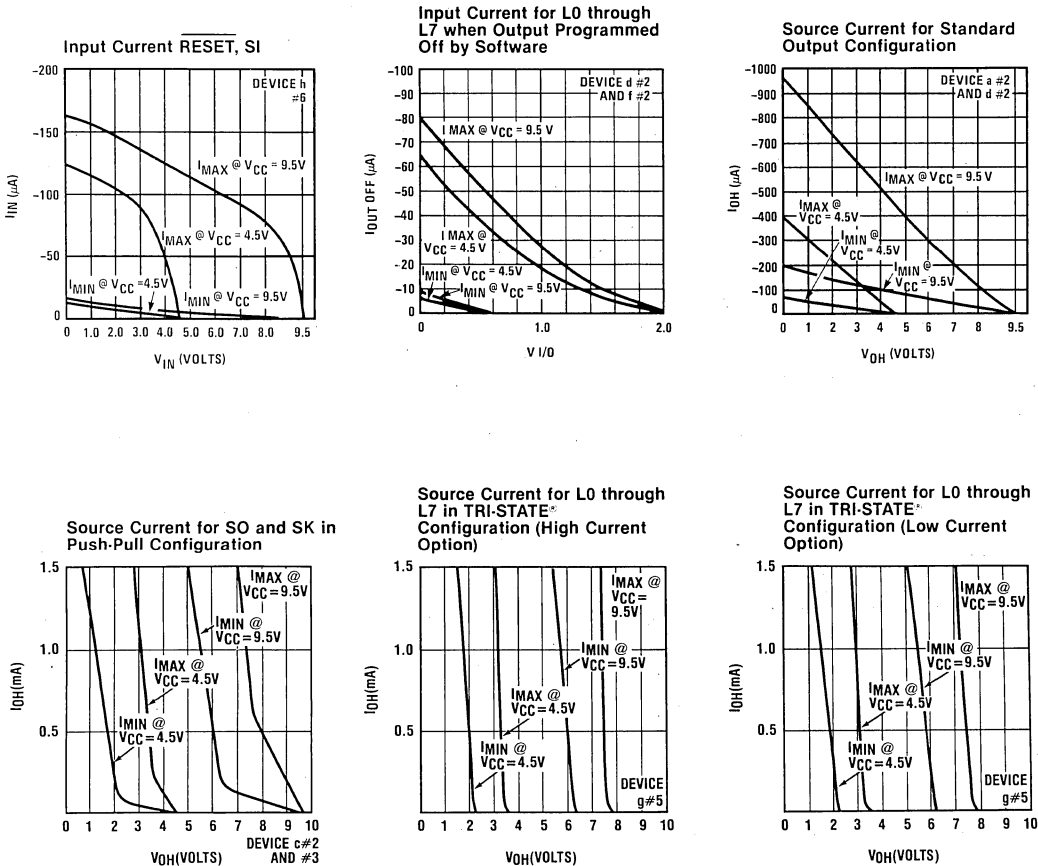


Figure 8a. COP410L/COP411L I/O DC Current Characteristics



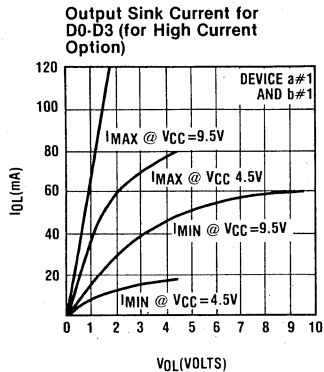
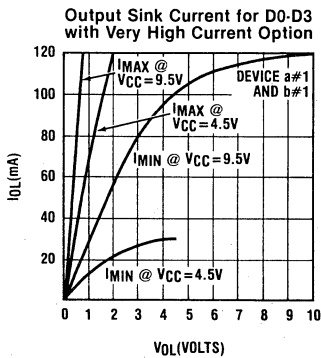
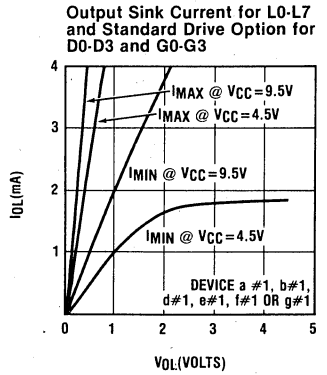
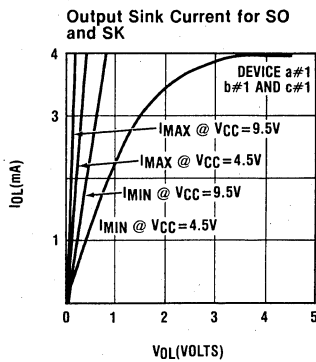
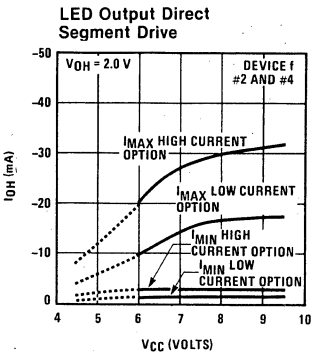
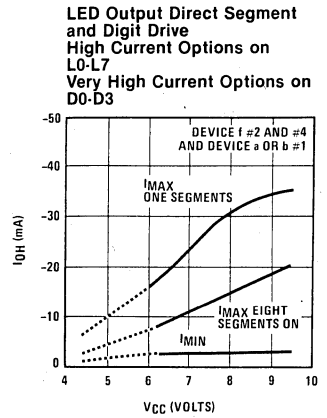
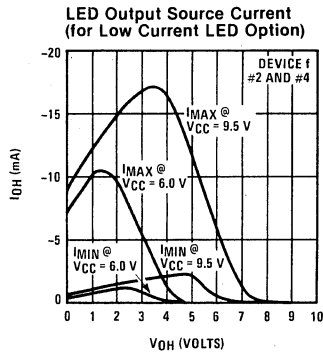
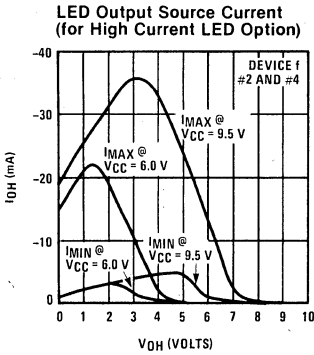


Figure 8a. COP410L/COP411L I/O DC Current Characteristics (continued)

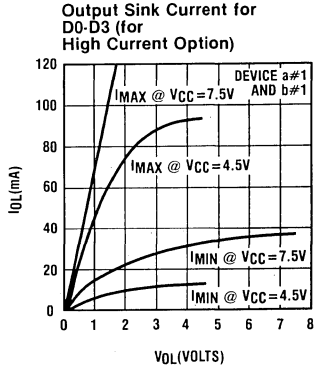
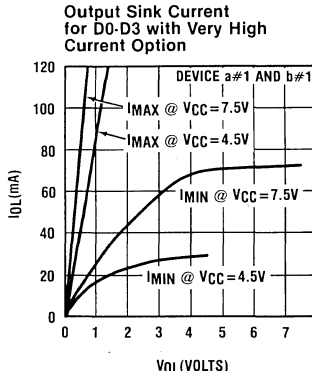
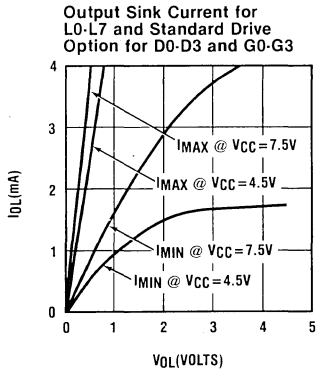
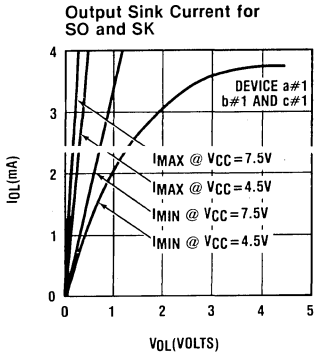
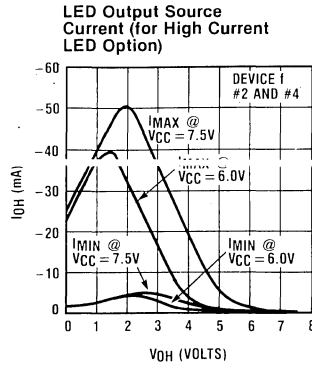
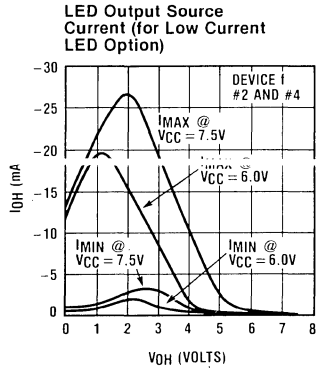
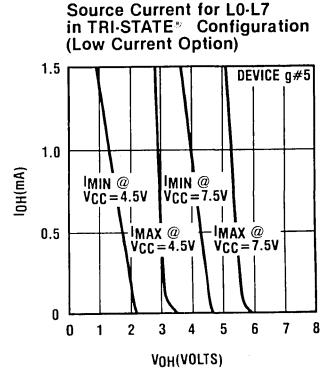
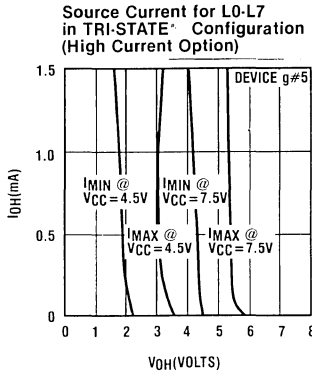
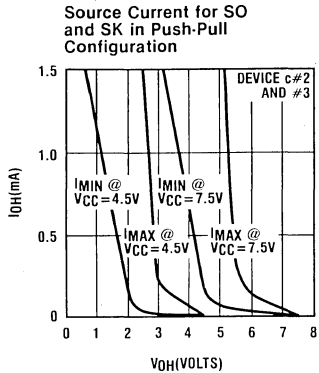
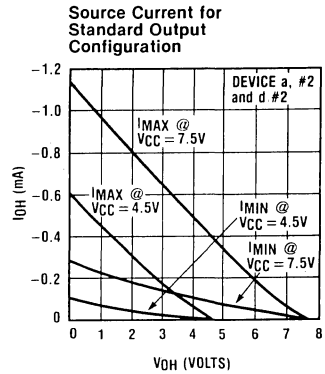
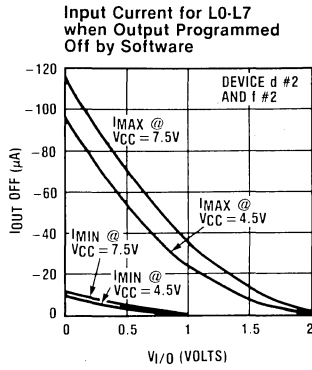
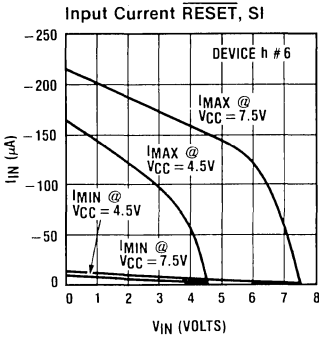


Figure 8b. COP310L/COP311L Input/Output Characteristics



Table 3. COP410L/411L Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
TRANSFER OF CONTROL INSTRUCTIONS						
JID		FF	<u>1 1 1 1</u>   <u>1 1 1 1</u>	ROM (PC <sub>8,A,M</sub> ) → PC <sub>7:0</sub>	None	Jump Indirect (Note 2)
JMP	a	6- --	<u>0 1 1 0</u>   <u>0 0 0</u>  a <sub>8</sub> a <sub>7:0</sub>	a → PC	None	Jump
JP	a	--	<u>1</u>   a <sub>6:0</sub> (pages 2,3 only)	a → PC <sub>6:0</sub>	None	Jump within Page (Note 3)
			or <u>1 1</u>   a <sub>5:0</sub> (all other pages)	a → PC <sub>5:0</sub>		
JSRP	a	--	<u>1 0</u>   a <sub>5:0</sub>	PC + 1 → SA → SB 010 → PC <sub>8:6</sub> a → PC <sub>5:0</sub>	None	Jump to Subroutine Page (Note 4)
JSR	a	6- --	<u>0 1 1 0</u>   <u>1 0 0</u>  a <sub>8</sub> a <sub>7:0</sub>	PC + 1 → SA → SB a → PC	None	Jump to Subroutine
RET		48	<u>0 1 0 0</u>   <u>1 0 0 0</u>	SB → SA → PC	None	Return from Subroutine
RETSK		49	<u>0 1 0 0</u>   <u>1 0 0 1</u>	SB → SA → PC	Always Skip on Return	Return from Subroutine then Skip
MEMORY REFERENCE INSTRUCTIONS						
CAMQ		33 3C	<u>0 0 1 1</u>   <u>0 0 1 1</u> <u>0 0 1 1</u>   <u>1 1 0 0</u>	A → Q <sub>7:4</sub> RAM(B) → Q <sub>3:0</sub>	None	Copy A, RAM to Q
LD	r	-5	<u>0 0</u>  r  <u>0 1 0 1</u>	RAM(B) → A Br ⊕ r → Br	None	Load RAM into A, Exclusive-OR Br with r
LQID		BF	<u>1 0 1 1</u>   <u>1 1 1 1</u>	ROM(PC <sub>8,A,M</sub> ) → Q SA → SB	None	Load Q Indirect (Note 2)
RMB	0	4C	<u>0 1 0 0</u>   <u>1 1 0 0</u>	0 → RAM(B) <sub>0</sub>	None	Reset RAM Bit
	1	45	<u>0 1 0 0</u>   <u>0 1 0 1</u>	0 → RAM(B) <sub>1</sub>		
	2	42	<u>0 1 0 0</u>   <u>0 0 1 0</u>	0 → RAM(B) <sub>2</sub>		
	3	43	<u>0 1 0 0</u>   <u>0 0 1 1</u>	0 → RAM(B) <sub>3</sub>		
SMB	0	4D	<u>0 1 0 0</u>   <u>1 1 0 1</u>	1 → RAM(B) <sub>0</sub>	None	Set RAM Bit
	1	47	<u>0 1 0 0</u>   <u>0 1 1 1</u>	1 → RAM(B) <sub>1</sub>		
	2	46	<u>0 1 0 0</u>   <u>0 1 1 0</u>	1 → RAM(B) <sub>2</sub>		
	3	4B	<u>0 1 0 0</u>   <u>1 0 1 1</u>	1 → RAM(B) <sub>3</sub>		
STII	y	7-	<u>0 1 1 1</u>   y	y → RAM(B) Bd + 1 → Bd	None	Store Memory Immediate and Increment Bd
X	r	-6	<u>0 0</u>  r  <u>0 1 1 0</u>	RAM(B) ↔ A Br ⊕ r → Br	None	Exchange RAM with A, Exclusive-OR Br with r
XAD	3,15	23	<u>0 0 1 0</u>   <u>0 0 1 1</u>	RAM(3,15) ↔ A	None	Exchange A with RAM (3,15)
		BF	<u>1 0 1 1</u>   <u>1 1 1 1</u>			
XDS	r	-7	<u>0 0</u>  r  <u>0 1 1 1</u>	RAM(B) ↔ A Bd - 1 → Bd Br ⊕ r → Br	Bd decrements past 0	Exchange RAM with A and Decrement Bd, Exclusive-OR Br with r
XIS	r	-4	<u>0 0</u>  r  <u>0 1 0 0</u>	RAM(B) ↔ A Bd + 1 → Bd Br ⊕ r → Br	Bd increments past 15	Exchange RAM with A and Increment Bd, Exclusive-OR Br with r

Table 3. COP410L/411L Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
REGISTER REFERENCE INSTRUCTIONS						
CAB		50	<u>0 1 0 1</u>   <u>0 0 0 0</u>	A → Bd	None	Copy A to Bd
CBA		4E	<u>0 1 0 0</u>   <u>1 1 1 0</u>	Bd → A	None	Copy Bd to A
LBI	r,d	--	<u>0 0</u>  r (d-1) (d = 0, 9:15)	r,d → B	Skip until not a LBI	Load B Immediate with r,d (Note 5)
LEI	y	33 6-	<u>0 0 1 1</u>   <u>0 0 1 1</u> <u>0 1 1 0</u>  y	y → EN	None	Load EN Immediate (Note 6)
TEST INSTRUCTIONS						
SKC		20	<u>0 0 1 0</u>   <u>0 0 0 0</u>		C = "1"	Skip if C is True
SKE		21	<u>0 0 1 0</u>   <u>0 0 0 1</u>		A = RAM(B)	Skip if A Equals RAM
SKGZ		33 21	<u>0 0 1 1</u>   <u>0 0 1 1</u> <u>0 0 1 0</u>   <u>0 0 0 1</u>		G <sub>3:0</sub> = 0	Skip if G is Zero (all 4 bits)
SKGBZ		33	<u>0 0 1 1</u>   <u>0 0 1 1</u>	1st byte		Skip if G Bit is Zero
	0	01	<u>0 0 0 0</u>   <u>0 0 0 1</u>	} 2nd byte	G <sub>0</sub> = 0	
	1	11	<u>0 0 0 1</u>   <u>0 0 0 1</u>		G <sub>1</sub> = 0	
	2	03	<u>0 0 0 0</u>   <u>0 0 1 1</u>		G <sub>2</sub> = 0	
	3	13	<u>0 0 0 1</u>   <u>0 0 1 1</u>		G <sub>3</sub> = 0	
SKMBZ	0	01	<u>0 0 0 0</u>   <u>0 0 0 1</u>		RAM(B) <sub>0</sub> = 0	Skip if RAM Bit is Zero
	1	11	<u>0 0 0 1</u>   <u>0 0 0 1</u>		RAM(B) <sub>1</sub> = 0	
	2	03	<u>0 0 0 0</u>   <u>0 0 1 1</u>		RAM(B) <sub>2</sub> = 0	
	3	13	<u>0 0 0 1</u>   <u>0 0 1 1</u>		RAM(B) <sub>3</sub> = 0	
INPUT/OUTPUT INSTRUCTIONS						
ING		33 2A	<u>0 0 1 1</u>   <u>0 0 1 1</u> <u>0 0 1 0</u>   <u>1 0 1 0</u>	G → A	None	Input G Ports to A
INL		33 2E	<u>0 0 1 1</u>   <u>0 0 1 1</u> <u>0 0 1 0</u>   <u>1 1 1 0</u>	L <sub>7:4</sub> → RAM(B) L <sub>3:0</sub> → A	None	Input L Ports to RAM, A
OBD		33 3E	<u>0 0 1 1</u>   <u>0 0 1 1</u> <u>0 0 1 1</u>   <u>1 1 1 0</u>	Bd → D	None	Output Bd to D Outputs
OMG		33 3A	<u>0 0 1 1</u>   <u>0 0 1 1</u> <u>0 0 1 1</u>   <u>1 0 1 0</u>	RAM(B) → G	None	Output RAM to G Ports
XAS		4F	<u>0 1 0 0</u>   <u>1 1 1 1</u>	A ↔ SIO, C → SKL	None	Exchange A with SIO (Note 2)

**Note 1:** All subscripts for alphabetical symbols indicate bit numbers unless explicitly defined (e.g., Br and Bd are explicitly defined). Bits are numbered 0 to N where 0 signifies the least significant bit (low-order, right-most bit). For example, A<sub>3</sub> indicates the most significant (left-most) bit of the 4-bit A register.

**Note 2:** For additional information on the operation of the XAS, JID, and LQID instructions, see below.

**Note 3:** The JP instruction allows a jump, while in subroutine pages 2 or 3, to any ROM location within the two-page boundary of pages 2 or 3. The JP instruction, otherwise, permits a jump to a ROM location within the current 64-word page. JP may not jump to the last word of a page.

**Note 4:** A JSRP transfers program control to subroutine page 2 (0010 is loaded into the upper 4 bits of P). A JSRP may not be used when in pages 2 or 3. JSRP may not jump to the last word in page 2.

**Note 5:** The machine code for the lower 4 bits of the LBI instruction equals the binary value of the "d" data *minus 1*, e.g., to load the lower four bits of B (Bd) with the value 9 (1001<sub>2</sub>), the lower 4 bits of the LBI instruction equal 8 (1000<sub>2</sub>). To load 0, the lower 4 bits of the LBI instruction should equal 15 (1111<sub>2</sub>).

**Note 6:** Machine code for operand field y for LEI instruction should equal the binary value to be latched into EN, where a "1" or "0" in each bit of EN corresponds with the selection or deselection of a particular function associated with each bit. (See Functional Description, EN Register.)

The following information is provided to assist the user in understanding the operation of several unique instructions and to provide notes useful to programmers in writing COP410L/411L programs.

### XAS Instruction

XAS (Exchange A with SIO) exchanges the 4-bit contents of the accumulator with the 4-bit contents of the SIO register. The contents of SIO will contain serial-in/serial-out shift register or binary counter data, depending on the value of the EN register. An XAS instruction will also affect the SK output. (See Functional Description, EN Register, above.) If SIO is selected as a shift register, an XAS instruction must be performed once every 4 instruction cycles to effect a continuous data stream.

### JID Instruction

JID (Jump Indirect) is an indirect addressing instruction, transferring program control to a new ROM location pointed to indirectly by A and M. It loads the lower 8 bits of the ROM address register PC with the contents of ROM addressed by the 9-bit word,  $PC_8$ , A, M.  $PC_8$  is not affected by this instruction.

Note that JID requires 2 instruction cycles to execute.

### LQID Instruction

LQID (Load Q Indirect) loads the 8-bit Q register with the contents of ROM pointed to by the 9-bit word  $PC_8$ , A, M. LQID can be used for table lookup or code conversion such as BCD to seven-segment. The LQID instruction "pushes" the stack ( $PC + 1 \rightarrow SA \rightarrow SB$ ) and replaces the least significant 8 bits of PC as follows:  $A \rightarrow PC_{7:4}$ ,  $RAM(B) \rightarrow PC_{3:0}$ , leaving  $PC_8$  unchanged. The ROM data pointed to by the new address is fetched and loaded into the Q latches. Next, the stack is "popped" ( $SB \rightarrow SA \rightarrow PC$ ), restoring the saved value of PC to continue sequential program execution. Since LQID pushes  $SA \rightarrow SB$ , the previous contents of SB are lost. Also, when LQID pops the stack, the previously pushed contents of SA are left in SB. The net result is that the contents of SA are placed in SB ( $SA \rightarrow SB$ ). Note that LQID takes two instruction cycle times to execute.

### Instruction Set Notes

- The first word of a COP410L/411L program (ROM address 0) must be a CLRA (Clear A) instruction.
- Although skipped instructions are not executed, one instruction cycle time is devoted to skipping each byte of the skipped instruction. Thus all program paths except JID and LQID take the same number of cycle times whether instructions are skipped or executed. JID and LQID instructions take 2 cycles if executed and 1 cycle if skipped.
- The ROM is organized into 8 pages of 64 words each. The Program Counter is a 9-bit binary counter, and will count through page boundaries. If a JP, JSRP, JID or LQID instruction is located in the last word of a page, the instruction operates as if it were in the next page. For example: a JP located in the last word of a page will jump to a location in the next page. Also, a LQID or JID located in the last word of page 3 or 7 will access data in the next group of 4 pages.

### Option List

The COP410L/411L mask-programmable options are assigned numbers which correspond with the COP410L pins.

The following is a list of COP410L options. When specifying a COP411L chip, Option 2 must be set to 3, Options 20, 21, and 22 to 0. The options are programmed at the same time as the ROM pattern to provide the user with the hardware flexibility to interface to various I/O components using little or no external circuitry.

Option 1 = 0: Ground Pin — no options available

Option 2: CKO Output (no option available for COP411L)  
 = 0: Clock output to ceramic resonator  
 = 1: Pin is RAM power supply ( $V_R$ ) input  
 = 2: Multi-COP SYNC input  
 = 3: No connection

Option 3: CKI Input  
 = 0: Oscillator input divided by 8 (500kHz max.)  
 = 1: Single-pin RC controlled oscillator divided by 4  
 = 2: External Schmitt trigger level clock divided by 4

Option 4:  $\overline{RESET}$  Input  
 = 0: Load device to  $V_{CC}$   
 = 1: Hi-Z input

Option 5:  $L_7$  Driver  
 = 0: Standard output  
 = 1: Open-drain output  
 = 2: High current LED direct segment drive output  
 = 3: High current TRI-STATE® push-pull output  
 = 4: Low-current LED direct segment drive output  
 = 5: Low-current TRI-STATE® push-pull output

Option 6:  $L_6$  Driver  
 same as Option 5

Option 7:  $L_5$  Driver  
 same as Option 5

Option 8:  $L_4$  Driver  
 same as Option 5

Option 9:  $V_{CC}$  Pin  
 = 0: 4.5V to 6.3V operation  
 = 1: 4.5V to 9.5V operation

Option 10:  $L_3$  Driver  
 same as Option 5

Option 11:  $L_2$  Driver  
 same as Option 5

Option 12:  $L_1$  Driver  
 same as Option 5

Option 13:  $L_0$  Driver  
 same as Option 5

Option 14: SI Input  
 = 0: load device to  $V_{CC}$   
 = 1: Hi-Z input

Option 15: SO Driver  
 = 0: Standard Output  
 = 1: Open-drain output  
 = 2: Push-pull output

Option 16: SK Driver  
 same as Option 15

Option 17: G<sub>0</sub> I/O Port  
= 0: Standard output  
= 1: Open-drain output

Option 18: G<sub>1</sub> I/O Port  
same as Option 17

Option 19: G<sub>2</sub> I/O Port  
same as Option 17

Option 20: G<sub>3</sub> I/O Port (no option available for COP411L)  
same as Option 17

Option 21: D<sub>3</sub> Output (no option available for COP411L)  
= 0: Very-high sink current standard output  
= 1: Very-high sink current open-drain output  
= 2: High sink current standard output  
= 3: High sink current open-drain output  
= 4: Standard LSTTL output (fanout = 1)  
= 5: Open-drain LSTTL output (fanout = 1)

Option 22: D<sub>2</sub> Output (no option available for COP411L)  
same as Option 21

Option 23: D<sub>1</sub> Output  
same as Option 21

Option 24: D<sub>0</sub> Output  
same as Option 21

Option 25: L Input Levels  
= 0: Standard TTL input levels ("0" = 0.8V, "1" = 2.0V)  
= 1: Higher voltage input levels ("0" = 1.2V, "1" = 3.6V)

Option 26: G Input Levels  
same as Option 25

Option 27: SI Input Levels  
same as Option 25

Option 28: COP Bonding  
= 0: COP410L (24-pin device)  
= 1: COP411L (20-pin device)  
= 2: Both 24- and 20-pin versions

#### Test Mode (Non-Standard Operation)

The SO output has been configured to provide for standard test procedures for the custom-programmed COP410L. With SO forced to logic "1", two test modes are provided, depending upon the value of SI:

- a. RAM and Internal Logic Test Mode (SI = 1)
- b. ROM Test Mode (SI = 0)

These special test modes should not be employed by the user; they are intended for manufacturing test only.

# COP420/COP421/COP422 and COP320/COP321/COP322 Single-Chip N-Channel Microcontrollers

## General Description

The COP420, COP421, COP422, COP320, COP321 and COP322 Single-Chip N-Channel Microcontrollers are members of the COPS™ family, fabricated using N-channel, silicon gate MOS technology. They are complete microcomputers containing all system timing, internal logic, ROM, RAM and I/O necessary to implement dedicated control functions in a variety of applications. Features include single supply operation, a variety of output configuration options, with an instruction set, internal architecture and I/O scheme designed to facilitate keyboard input, display output and BCD data manipulation. The COP421 is identical to the COP420, except with 19 I/O lines instead of 23; the COP422 has 15 I/O lines. They are an appropriate choice for use in numerous human interface control environments. Standard test procedures and reliable high-density fabrication techniques provide the medium to large volume customers with a customized Controller Oriented Processor at a low end-product cost.

The COP320 is the extended temperature range version of the COP420 (likewise the COP321 and COP322 are the extended temperature range versions of the COP421/COP422). The COP320/321/322 are exact functional equivalents of the COP420/421/422.

## Features

- Low cost
- Powerful instruction set
- 1k × 8 ROM, 64 × 4 RAM
- 23 I/O lines (COP420, COP320)
- True vectored interrupt, plus restart
- Three-level subroutine stack
- 4.0 μs instruction time
- Single supply operation
- Internal time-base counter for real-time processing
- Internal binary counter register with MICROWIRE™ compatible serial I/O capability
- General purpose and TRI-STATE® outputs
- TTL/CMOS compatible in and out
- LED direct drive outputs
- MICROBUS™ compatible
- Software/hardware compatible with other members of COP400 family
- Extended temperature range device COP320/COP321/COP322 (-40°C to +85°C)

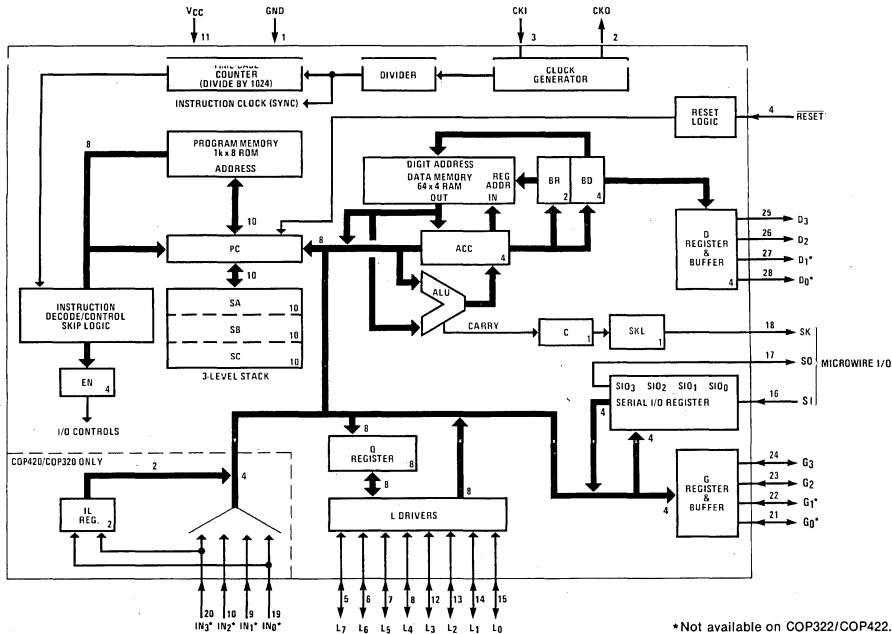


Figure 1. COP420/COP421/COP422, COP320/COP321/COP322 Block Diagram



## COP420/COP421/COP422 and COP320/COP321/COP322

### Absolute Maximum Ratings

Voltage at Any Pin	-0.3V to +7V	Package Power Dissipation 24 and 28 pin	750 mW at 25°C 400 mW at 70°C 250 mW at 85°C
Operating Temperature Range	0°C to 70°C		
COP420/COP421/COP422	0°C to 70°C		
COP320/COP321/COP322	-40°C to +85°C	Package Power Dissipation	650 mW at 25°C 300 mW at 70°C 200 mW at 85°C
Storage Temperature Range	-65°C to +150°C	20 pin	
Total Sink Current	75 mA		
Total Source Current	95 mA	Lead Temperature (soldering, 10 sec.)	300°C

*Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

### COP420/COP421/COP422

#### DC Electrical Characteristics $0^{\circ}\text{C} \leq T_A \leq 70^{\circ}\text{C}$ , $4.5\text{V} \leq V_{CC} \leq 6.3\text{V}$ unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Operation Voltage		4.5	6.3	V
Power Supply Ripple	Peak to Peak (Note 3)		0.4	V
Supply Current	Outputs Open		38	mA
Supply Current	Outputs Open, $V_{CC} = 5\text{V}$ , $T_A = 25^{\circ}\text{C}$		30	mA
Input Voltage Levels				
CKI Input Levels				
Crystal Input				
Logic High	$V_{CC} = \text{Max.}$	3.0		V
Logic High	$V_{CC} = 5\text{V} \pm 5\%$	2.0		V
Logic Low		-0.3	0.4	V
TTL Input	$V_{CC} = 5\text{V} \pm 5\%$			
Logic High		2.0		V
Logic Low		-0.3	0.8	V
Schmitt Trigger Inputs				
RESET, CKI ( $\div 4$ )				
Logic High		$0.7 V_{CC}$		V
Logic Low		-0.3	0.6	V
SO Input Level (Test Mode)		2.0	3.0	V
All Other Inputs				
Logic High	$V_{CC} = \text{Max.}$	3.0		V
Logic High	$V_{CC} = 5\text{V} \pm 5\%$	2.0		V
Logic Low		-0.3	0.8	V
Input Levels High Trip Option				
Logic High		3.6		V
Logic Low		-0.3	1.2	V
Input Load Source Current	$V_{CC} = 5\text{V}$ , $V_{IN} = 0\text{V}$			
CKO		-4	-800	$\mu\text{A}$
All Others		-100	-800	$\mu\text{A}$
Input Capacitance			7	pF
Hi-Z Input Leakage	$V_{CC} = 5\text{V}$	-1	+1	$\mu\text{A}$
Output Voltage levels				
Standard Outputs				
TTL Operation	$V_{CC} = 5\text{V} \pm 5\%$			
Logic High	$I_{OH} = -100\mu\text{A}$	2.4		V
Logic Low	$I_{OL} = 1.6\text{mA}$	-0.3	0.4	V
CMOS Operation				
Logic High	$I_{OH} = -10\mu\text{A}$	$V_{CC} - 1$		V

**COP420/COP421/COP422****DC Electrical Characteristics** (Cont'd)  $0^{\circ}\text{C} \leq T_A \leq 70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 6.3\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Output Current Levels				
LED Direct Drive Output	$V_{CC} = 6\text{V}$			
Logic High	$V_{OH} = 2.0\text{V}$	2.5	14	mA
CKI Sink Current (R/C Option)	$V_{IN} = 3.5\text{V}$	2		mA
CKO (RAM Supply Current)	$V_R = 3.3\text{V}$		3	mA
TRI-STATE® or Open Drain Leakage Current	$V_{CC} = 5\text{V}$	-2.5	+2.5	$\mu\text{A}$
Output Current Levels				
Output Sink Current ( $I_{OL}$ )	$V_{CC} = 6.3\text{V}$ , $V_{OL} = 0.4\text{V}$	-2.0		mA
Output Source Current ( $I_{OH}$ )	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	-1.0		mA
Standard Configuration				
All Outputs	$V_{CC} = 6.3\text{V}$ , $V_{OH} = 3.0\text{V}$	-200	-900	$\mu\text{A}$
	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-100	-500	$\mu\text{A}$
Push-Pull Configuration				
SO, SK Outputs	$V_{CC} = 6.3\text{V}$ , $V_{OH} = 3.0\text{V}$	-1.0		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-0.4		mA
TRI-STATE Configuration				
L <sub>0</sub> -L <sub>7</sub> Outputs	$V_{CC} = 6.3\text{V}$ , $V_{OH} = 3.0\text{V}$	-2.0		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-0.8		mA
LED Configuration				
L <sub>0</sub> -L <sub>7</sub> Outputs	$V_{CC} = 6.3\text{V}$ , $V_{OH} = 3.0\text{V}$	-1.0		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-0.5		mA
Allowable Sink Current				
Per Pin (L, D, G)			10	mA
Per Pin (All Others)			2	mA
Per Port (L)			16	mA
Per Port (D, G)			10	mA
Allowable Source Current				
Per Pin (L)			-15	mA
Per Pin (All Others)			-1.5	mA

**COP320/COP321/COP322****DC Electrical Characteristics**-40°C ≤ T<sub>A</sub> ≤ +85°C, 4.5V ≤ V<sub>CC</sub> ≤ 5.5V unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Operation Voltage		4.5	5.5	V
Power Supply Ripple	Peak to Peak (Note 3)		0.4	V
Supply Current	T <sub>A</sub> = -40°C, Outputs Open		40	mA
Input Voltage Levels				
CKI Input Levels				
Crystal Input				
Logic High		2.2		V
Logic Low		-0.3	0.3	V
TTL Input	V <sub>CC</sub> = 5V ± 5%			
Logic High		2.2		V
Logic Low		-0.3	0.6	V
Schmitt Trigger Inputs				
RESET, CKI (÷4)				
Logic High		0.7V <sub>CC</sub>		V
Logic Low		-0.3	0.4	V
SO Input Level (Test Mode)		2.0	3.0	V
All Other Inputs				
Logic High	V <sub>CC</sub> = Max.	3.0		V
Logic High	V <sub>CC</sub> = 5V ± 5%	2.2		V
Logic Low		-0.3	0.6	V
Input Levels High Trip Option				
Logic High		3.6		V
Logic Low		-0.3	1.2	V
Input Load Source Current	V <sub>CC</sub> = 5V, V <sub>IN</sub> = 0V			
CKO		-4	-800	μA
All Others		-100	-800	μA
Input Capacitance			7	pF
Hi-Z Input Leakage	V <sub>CC</sub> = 5V	-2	+2	μA
Output Voltage levels				
Standard Outputs				
TTL Operation	V <sub>CC</sub> = 5V ± 5%			
Logic High	I <sub>OH</sub> = -75μA	2.4		V
Logic Low	I <sub>OL</sub> = 1.6mA	-0.3	0.4	V
CMOS Operation				
Logic High	I <sub>OH</sub> = -10μA	V <sub>CC</sub> - 1		V
Logic Low	I <sub>OL</sub> = 10μA	-0.3	0.2	V
Output Current Levels				
LED Direct Drive Output	V <sub>CC</sub> = 5V (Note 4)			
Logic High	V <sub>OH</sub> = 2.0V	1.0	12	mA
CKI Sink Current (R/C Option)	V <sub>IN</sub> = 3.5V	2		mA
CKO (RAM Supply Current)	V <sub>R</sub> = 3.3V		4	mA
TRI-STATE® or Open Drain Leakage Current	V <sub>CC</sub> = 5V	-5	+5	μA
Allowable Sink Current				
Per Pin (L, D, G)			10	mA
Per Pin (All Others)			2	mA
Per Port (L)			16	mA
Per Port (D, G)			10	mA
Allowable Source Current				
Per Pin (L)			-15	mA
Per Pin (All Others)			-1.5	mA

## AC Electrical Characteristics

COP420/COP421/COP422  $0^{\circ}\text{C} \leq T_A \leq 70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 6.3\text{V}$  unless otherwise noted.

COP320/COP321/COP322  $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 5.5\text{V}$  unless otherwise noted.

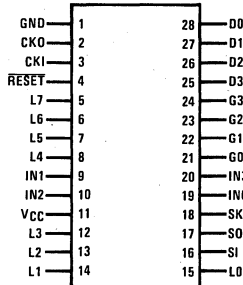
Parameter	Conditions	Min.	Max.	Units
Instruction Cycle Time		4	10	$\mu\text{s}$
Operating CKI Frequency	$\div 16$ mode $\div 8$ mode	1.6 0.8	4.0 2.0	MHz MHz
CKI Duty Cycle (Note 1)		40	60	%
Rise Time	Freq. = 4 MHz		60	ns
Fall Time	Freq. = 4 MHz		40	ns
CKI Using RC (Figure 8c)	$\div 4$ mode			
Frequency	$R = 15\text{k}\Omega \pm 5\%$ , $C = 100\text{pF} \pm 10\%$	0.5	1.0	MHz
Instruction Cycle Time		4	8	$\mu\text{s}$
CKO as SYNC input (Figure 8d)				
$t_{\text{SYNC}}$	Figure 3a	50		ns
Inputs:				
SI				
$t_{\text{SETUP}}$		0.3		$\mu\text{s}$
$t_{\text{HOLD}}$		250		ns
All Other Inputs				
$t_{\text{SETUP}}$		1.7		$\mu\text{s}$
$t_{\text{HOLD}}$		300		ns
Output Propagation Delay	Test Conditions: $R_L = 5\text{k}\Omega$ , $C_L = 50\text{pF}$ , $V_{\text{OUT}} = 1.5\text{V}$	300		ns
SO and SK				
$t_{\text{pd1}}$			1.0	$\mu\text{s}$
$t_{\text{pd0}}$			1.0	$\mu\text{s}$
CKO				
$t_{\text{pd1}}$			0.25	$\mu\text{s}$
$t_{\text{pd0}}$			0.25	$\mu\text{s}$
All Other Outputs				
$t_{\text{pd1}}$			1.4	$\mu\text{s}$
$t_{\text{pd0}}$			1.4	$\mu\text{s}$
MICROBUS™ Timing	$C_i = 100\text{pF}$ , $V_{CC} = 5\text{V} \pm 5\%$			
Read Operation (Figure 4)				
Chip Select Stable before $\overline{\text{RD}}$ — $t_{\text{CSR}}$		65		ns
Chip Select Hold Time for $\overline{\text{RD}}$ — $t_{\text{RCS}}$		20		ns
$\overline{\text{RD}}$ Pulse Width— $t_{\text{RR}}$		400		ns
Data Delay from $\overline{\text{RD}}$ — $t_{\text{RD}}$			375	ns
$\overline{\text{RD}}$ to Data Floating— $t_{\text{DF}}$			250	ns
Write Operation (Figure 5)				
Chip Select Stable before $\overline{\text{WR}}$ — $t_{\text{CSW}}$		65		ns
Chip Select Hold Time for $\overline{\text{WR}}$ — $t_{\text{WCS}}$		20		ns
$\overline{\text{WR}}$ Pulse Width— $t_{\text{WW}}$		400		ns
Data Set-Up Time for $\overline{\text{WR}}$ — $t_{\text{DW}}$		320		ns
Data Hold Time for $\overline{\text{WR}}$ — $t_{\text{WD}}$		100		ns
INTR Transition Time from $\overline{\text{WR}}$ — $t_{\text{WI}}$			700	ns

**Note 1:** Duty cycle =  $t_{W1}/(t_{W1} + t_{W0})$ .

**Note 2:** See Figure 9 for additional I/O characteristics.

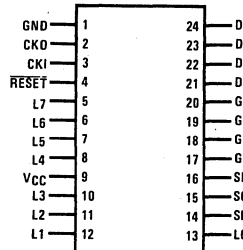
**Note 3:** Voltage change must be less than 0.5 volts in a 1 ms period.

**Note 4:** Exercise great care not to exceed maximum device power dissipation limits when direct driving LEDs (or sourcing similar loads) at high temperature.



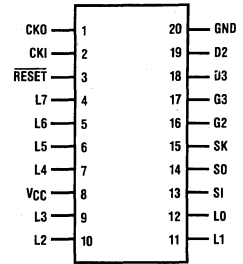
COP420, COP320

Order Number COP420N, COP320N  
NS Package N28A



COP421, COP321

Order Number COP421N, COP 321N  
NS Package N24A



COP422, COP322

Order Number COP422N, COP322N  
NS Package N20A

Figure 2. Connection Diagrams

Pin	Description	Pin	Description
L7-L0	8 bidirectional I/O ports with TRI-STATE®	SK	Logic-controlled clock (or general purpose output)
G3-G0	4 bidirectional I/O ports	CKI	System oscillator input
D3-D0	4 general purpose outputs	CKO	System oscillator output (or general purpose input or RAM power supply)
IN3-IN0	4 general purpose inputs (COP420/320 only)	RESET	System reset input
SI	Serial input (or counter input)	VCC	Power supply
SO	Serial output (or general purpose output)	GND	Ground

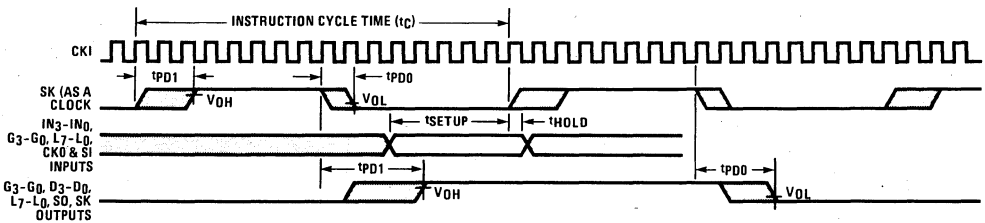


Figure 3. Input/Output Timing Diagrams (crystal divide by 16 mode)

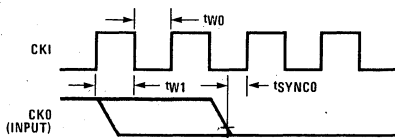


Figure 3A. Synchronization Timing

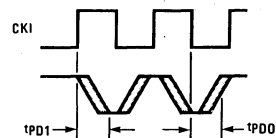


Figure 3B. CKO Output Timing

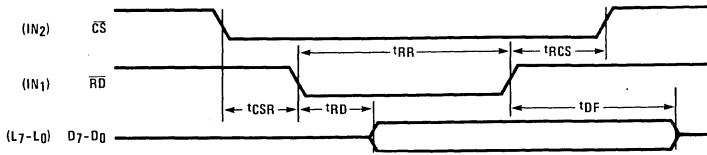


Figure 4. MICROBUS™ Read Operation Timing

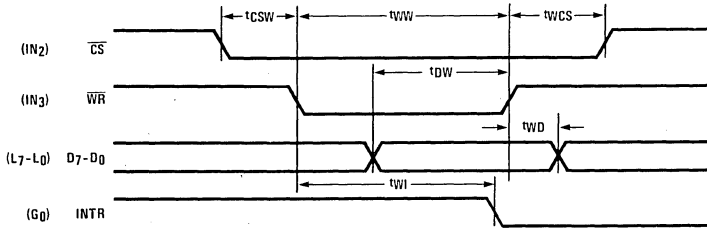


Figure 5. MICROBUS™ Write Operation Timing

## Functional Description COP420/COP421/COP422, COP320/COP321/COP322

For ease of reading this description, only COP420 and/or COP421 are referenced; however, all such references apply equally to the COP422, COP322, COP320 and/or COP321, respectively.

A block diagram of the COP420 is given in figure 1. Data paths are illustrated in simplified form to depict how the various logic elements communicate with each other in implementing the instruction set of the device. Positive logic is used. When a bit is set, it is a logic "1" (greater than 2 volts). When a bit is reset, it is a logic "0" (less than 0.8 volts).

### Program Memory

Program Memory consists of a 1,024 byte ROM. As can be seen by an examination of the COP420/421 instruction set, these words may be program instructions, program data or ROM addressing data. Because of the special characteristics associated with the JP, JSRP, JID and LQID instructions, ROM must often be thought of as being organized into 16 pages of 64 words each.

ROM addressing is accomplished by a 10-bit PC register. Its binary value selects one of the 1,024 8-bit words contained in ROM. A new address is loaded into the PC register during each instruction cycle. Unless the instruction is a transfer of control instruction, the PC register is loaded with the next sequential 10-bit binary count value. Three levels of subroutine nesting are implemented by the 10-bit subroutine save registers, SA, SB and SC, providing a last-in, first-out (LIFO) hardware subroutine stack.

ROM instruction words are fetched, decoded and executed by the Instruction Decode, Control and Skip Logic circuitry.

### Data Memory

Data memory consists of a 256-bit RAM, organized as 4 data registers of 16 4-bit digits. RAM addressing is implemented by a 6-bit **B register** whose upper 2 bits (Br) select 1 of 4 data registers and lower 4 bits (Bd) select 1

of 16 4-bit digits in the selected data register. While the 4-bit contents of the selected RAM digit (M) is usually loaded into or from, or exchanged with, the A register (accumulator), it may also be loaded into or from the Q latches or loaded from the L ports. RAM addressing may also be performed directly by the LDD and XAD instructions based upon the 6-bit contents of the operand field of these instructions. The Bd register also serves as a source register for 4-bit data sent directly to the D outputs.

### Internal Logic

The 4-bit **A register** (accumulator) is the source and destination register for most I/O, arithmetic, logic and data memory access operations. It can also be used to load the Br and Bd portions of the B register, to load and input 4 bits of the 8-bit Q latch data, to input 4 bits of the 8-bit L I/O port data and to perform data exchanges with the SIO register.

A **4-bit adder** performs the arithmetic and logic functions of the COP420/421, storing its results in A. It also outputs a carry bit to the 1-bit **C register**, most often employed to indicate arithmetic overflow. The C register, in conjunction with the XAS instruction and the EN register, also serves to control the SK output. C can be outputted directly to SK or can enable SK to be a sync clock each instruction cycle time. (See XAS instruction and EN register description, below.)

Four **general-purpose inputs**,  $IN_3 - IN_0$ , are provided;  $IN_1$ ,  $IN_2$  and  $IN_3$  may be selected, by a mask-programmable option, as Read Strobe, Chip Select and Write Strobe inputs, respectively, for use in MICROBUS™ applications.

The **D register** provides 4 general-purpose outputs and is used as the destination register for the 4-bit contents of Bd.

The **G register** contents are outputs to 4 general-purpose bidirectional I/O ports.  $G_0$  may be mask-programmed as an output for MICROBUS™ applications.

The **Q register** is an internal, latched, 8-bit register, used to hold data loaded to or from M and A, as well as 8-bit data from ROM. Its contents are output to the L I/O ports when the L drivers are enabled under program control. (See LEI instruction). With the MICROBUS™ option selected, Q can also be loaded with the 8-bit contents of the L I/O ports upon the occurrence of a write strobe from the host CPU.

The **8 L drivers**, when enabled, output the contents of latched Q data to the L I/O ports. Also, the contents of L may be read directly into A and M. As explained above, the MICROBUS™ option allows L I/O port data to be latched into the Q register. L I/O ports can be directly connected to the segments of a multiplexed LED display (using the LED Direct Drive output configuration option) with Q data being outputted to the Sa–Sg and decimal point segments of the display.

The **SIO register** functions as a 4-bit serial-in/serial-out shift register or as a binary counter depending on the contents of the EN register. (See EN register description, below.) Its contents can be exchanged with A, allowing it to input or output a continuous serial data stream. SIO may also be used to provide additional parallel I/O by connecting SO to external serial-in/parallel-out shift registers. For example of additional parallel output capacity see **Application #2**.

The XAS instruction copies C into the **SKL latch**. In the counter mode, SK is the output of SKL; in the shift register mode, SK outputs SKL ANDed with the clock.

The **EN register** is an internal 4-bit register loaded under program control by the LEI instruction. The state of each bit of this register selects or deselects the particular feature associated with each bit of the EN register (EN<sub>3</sub>–EN<sub>0</sub>).

1. The least significant bit of the enable register, EN<sub>0</sub>, selects the SIO register as either a 4-bit shift register or a 4-bit binary counter. With EN<sub>0</sub> set, SIO is an asynchronous binary counter, *decrementing* its value by one upon each low-going pulse ("1" to "0") occurring on the SI input. Each pulse must be at least two instruction cycles wide. SK outputs the value of SKL. The SO output is equal to the value of EN<sub>3</sub>. With EN<sub>0</sub> reset, SIO is a serial shift register shifting left each instruction cycle time. The data present at SI goes into the least significant bit of SIO. SO can be enabled to output the most significant bit of SIO each cycle time. (See 4 below.) The SK output becomes a logic-controlled clock.
2. With EN<sub>1</sub> set the IN<sub>1</sub> input is enabled as an interrupt input. Immediately following an interrupt, EN<sub>1</sub> is reset to disable further interrupts.
3. With EN<sub>2</sub> set, the L drivers are enabled to output the data in Q to the L I/O ports. Resetting EN<sub>2</sub> disables the L drivers, placing the L I/O ports in a high-impedance input state.
4. EN<sub>3</sub>, in conjunction with EN<sub>0</sub>, affects the SO output. With EN<sub>0</sub> set (binary counter option selected) SO will output the value loaded into EN<sub>3</sub>. With EN<sub>0</sub> reset (serial shift register option selected), setting EN<sub>3</sub> enables SO as the output of the SIO shift register, outputting serial shifted data each instruction time. Resetting EN<sub>3</sub> with the serial shift register option selected disables SO as the shift register output; data continues to be shifted through SIO and can be exchanged with A via an XAS instruction but SO remains reset to "0." The table below provides a summary of the modes associated with EN<sub>3</sub> and EN<sub>0</sub>.

Enable Register Modes — Bits EN<sub>3</sub> and EN<sub>0</sub>

EN <sub>3</sub>	EN <sub>0</sub>	SIO	SI	SO	SK
0	0	Shift Register	Input to Shift Register	0	If SKL = 1, SK = CLOCK If SKL = 0, SK = 0
1	0	Shift Register	Input to Shift Register	Serial Out	If SKL = 1, SK = CLOCK If SKL = 0, SK = 0
0	1	Binary Counter	Input to Binary Counter	0	If SKL = 1, SK = 1 If SKL = 0, SK = 0
1	1	Binary Counter	Input to Binary Counter	1	If SKL = 1, SK = 1 If SKL = 0, SK = 0

## Interrupt

The following features are associated with the  $IN_1$  interrupt procedure and protocol and must be considered by the programmer when utilizing interrupts.

- The interrupt, once acknowledged as explained below, pushes the next sequential program counter address ( $PC + 1$ ) onto the stack, pushing in turn the contents of the other subroutine-save registers to the next lower level ( $PC + 1 \rightarrow SA \rightarrow SB \rightarrow SC$ ). Any previous contents of SC are lost. The program counter is set to hex address 0FF (the last word of page 3) and  $EN_1$  is reset.
- An interrupt will be acknowledged only after the following conditions are met:
  - $EN_1$  has been set.
  - A low-going pulse ("1" to "0") at least two instruction cycles wide occurs on the  $IN_1$  input.
  - A currently executing instruction has been completed.
  - All successive transfer of control instructions and successive LBIs have been completed (e.g., if the main program is executing a JP instruction which transfers program control to another JP instruction, the interrupt will not be acknowledged until the second JP instruction has been executed).
- Upon acknowledgement of an interrupt, the skip logic status is saved and later restored upon popping of the stack. For example, if an interrupt occurs during the execution of ASC (Add with Carry, Skip on Carry) instruction which results in carry, the skip logic status is saved and program control is transferred to the interrupt servicing routine at hex address 0FF. At the *end* of the interrupt routine, a RET instruction is executed to "pop" the stack and return program control to the instruction following the original ASC. *At this time*, the skip logic is enabled and skips this instruction because of the previous ASC carry. Subroutines and LDIR instructions should not be nested within the interrupt service routine, since their popping the stack will enable any previously saved main program skips, interfering with the orderly execution of the interrupt routine.
- The first instruction of the interrupt routine at hex address 0FF must be a NOP.
- A LEI instruction can be put immediately before the RET to re-enable interrupts.

## Microbus™ Interface

The COP420 has an option which allows it to be used as a peripheral microprocessor device, inputting and outputting data from and to a host microprocessor ( $\mu P$ ).  $IN_1$ ,  $IN_2$  and  $IN_3$  general purpose inputs become **MICROBUS™ compatible** read-strobe, chip-select, and write-strobe lines, respectively.  $IN_1$  becomes  $\overline{RD}$  — a logic "0" on this input will cause Q latch data to be enabled to the L ports for input to the  $\mu P$ .  $IN_2$  becomes CS — a logic "0" on this line selects the COP420 as the  $\mu P$  peripheral device by enabling the operation of the  $\overline{RD}$  and  $\overline{WR}$  lines and allows for the selection of one of several peripheral components.  $IN_3$  becomes  $\overline{WR}$  — a logic "0" on this line will write bus data from the L ports to the Q latches for input to the COP420.  $G_0$  becomes INTR a "ready" output, reset by a write pulse from the

$\mu P$  on the  $\overline{WR}$  line, providing the "handshaking capability necessary for asynchronous data transfer between the host CPU and the COP420.

This option has been designed for compatibility with National's MICROBUS™ — a standard interconnect system for 8-bit parallel data transfer between MOS/LSI CPUs and interfacing devices. (See MICROBUS™ National Publication.) The functioning and timing relationships between the COP420 signal lines affected by this option are as specified for the MICROBUS™ interface, and are given in the AC electrical characteristics and shown in the timing diagrams (figures 4 and 5). Connection of the COP420 to the MICROBUS™ is shown in Figure 6.

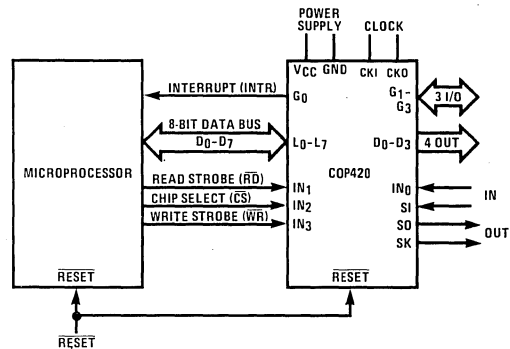


Figure 6. MICROBUS™ Option Interconnect

## Initialization

The Reset Logic, internal to the COP420/421, will initialize (clear) the device upon power-up if the power supply rise time is less than 1ms and greater than  $1\mu s$ . If the power supply rise time is greater than 1ms, the user must provide an external RC network and diode to the  $\overline{RESET}$  pin as shown below. The  $\overline{RESET}$  pin is configured as a Schmitt trigger input. If not used it should be connected to  $V_{CC}$ . Initialization will occur whenever a logic "0" is applied to the  $\overline{RESET}$  input, provided it stays low for at least three instruction cycle times.

Upon initialization, the PC register is cleared to 0 (ROM address 0) and the A, B, C, D, EN, and G registers are cleared. The SK output is enabled as a SYNC output, providing a pulse each instruction cycle time. *Data Memory (RAM) is not cleared upon initialization.* The first instruction at address 0 must be a CLRA.

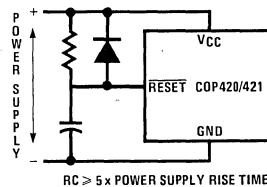


Figure 7. Power-Up Clear Circuit



### Oscillator

There are four basic clock oscillator configurations available as shown by figure 8.

- a. Crystal Controlled Oscillator.** CKI and CKO are connected to an external crystal. The instruction cycle time equals the crystal frequency divided by 16 (optional by 8).
- b. External Oscillator.** CKI is an external clock input signal. The external frequency is divided by 16 (optional by 8) to give the instruction cycle time. CKO is now available to be used as the RAM power supply ( $V_R$ ) or as a general purpose input.
- c. RC Controlled Oscillator.** CKI is configured as a single pin RC controlled Schmitt trigger oscillator. The instruction cycle equals the oscillation frequency divided by 4. CKO is available for non-timing functions.
- d. Externally Synchronized Oscillator.** Intended for use in multi-COP systems, CKO is programmed to function as an input connected to the SK output of another COP420/421 with CKI connected as shown. In this configuration, the SK output connected to CKO must provide a SYNC (instruction cycle) signal to CKO, thereby allowing synchronous data transfer between the COPs using only the SI and SO serial I/O pins in conjunction with the XAS instruction. Note that on power-up SK is automatically enabled as a SYNC output (See Functional Description, Initialization, above).

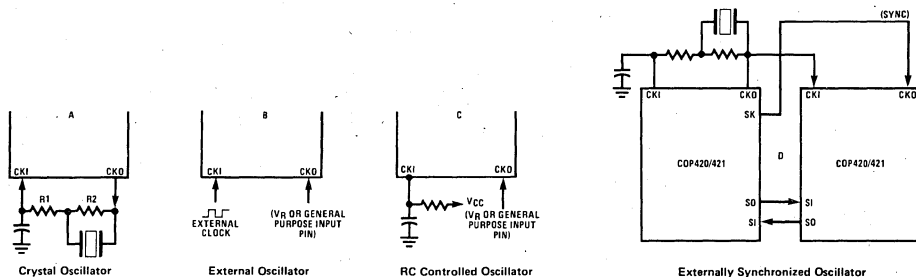
### CKO Pin Options

In a crystal controlled oscillator system, CKO is used as an output to the crystal network. As an option CKO can be a SYNC input as described above. As another option CKO can be a general purpose input, read into bit 2 of A (accumulator) upon execution of an INIL instruction. As another option, CKO can be a RAM power supply pin ( $V_R$ ), allowing its connection to a standby/backup power supply to maintain the integrity of RAM data with minimum power drain when the main supply is inoperative or shut down to conserve power. Using either option is appropriate in applications where the COP420/421 system timing configuration does not require use of the CKO pin.

### RAM Keep-Alive Option (Not available on COP422)

Selecting CKO as the RAM power supply ( $V_R$ ) allows the user to shut off the chip power supply ( $V_{CC}$ ) and maintain data in the RAM. To insure that RAM data integrity is maintained, the following conditions must be met:

1.  $\overline{RESET}$  must go low before  $V_{CC}$  goes below spec during power off;  $V_{CC}$  must be within spec before  $\overline{RESET}$  goes high on power up.
2.  $V_R$  must be within the operating range of the chip, and equal to  $V_{CC} \pm 1V$  during normal operation.
3.  $V_R$  must be  $\geq 3.3V$  with  $V_{CC}$  off.



**Crystal Oscillator**

Crystal Value	Component Values		
	R1 ( $\Omega$ )	R2 ( $\Omega$ )	C (pF)
4 MHz	1k	1M	27
3.58 MHz	1k	1M	27
2.09 MHz	1k	1M	56

**RC Controlled Oscillator**

R (k $\Omega$ )	C (pF)	Instruction Cycle Time
		( $\mu s$ )
12	100	$5 \pm 20\%$
6.8	220	$5.3 \pm 23\%$
8.2	300	$8 \pm 29\%$
22	100	$8.6 \pm 16\%$

Note:  $50 k\Omega \geq R \geq 5 k\Omega$   
 $360 pF \geq C \geq 50 pF$

Figure 8. COP420/421/COP320/321 Oscillator

**I/O Options**

COP420/421 outputs have the following optional configurations, illustrated in Figure 9a:

- a. **Standard** — an enhancement mode device to ground in conjunction with a depletion-mode device to  $V_{CC}$ , compatible with TTL and CMOS input requirements. Available on SO, SK, and all D and G outputs.
- b. **Open-Drain** — an enhancement-mode device to ground only, allowing external pull-up as required by the user's application. Available on SO, SK, and all D and G outputs.
- c. **Push-Pull** — An enhancement-mode device to ground in conjunction with a depletion-mode device paralleled by an enhancement-mode device to  $V_{CC}$ . This configuration has been provided to allow for fast rise and fall times when driving capacitive loads. Available on SO and SK outputs only.
- d. **Standard L** — same as a., but may be disabled. Available on L outputs only.
- e. **Open Drain L** — same as b., but may be disabled. Available on L outputs only.
- f. **LED Direct Drive** — an enhancement-mode device to ground and to  $V_{CC}$ , meeting the typical current sourcing requirements of the segments of an LED display. The sourcing device is clamped to limit current flow. These devices may be turned off under program control (See Functional Description, EN Register), placing the outputs in a high-impedance state to provide required LED segment blanking for a multiplexed display.
- g. **TRI-STATE® Push-Pull** — an enhancement-mode device to ground and  $V_{CC}$ . These outputs are TRI-STATE outputs, allowing for connection of these outputs to a data bus shared by other bus drivers.

COP420/COP421 inputs have the following optional configurations:

- h. An on-chip depletion load device to  $V_{CC}$ .
- i. A Hi-Z input which must be driven to a "1" or "0" by external components.

The above input and output configurations share common enhancement-mode and depletion-mode devices. Specifically, all configurations use one or more of six devices (numbered 1-6, respectively). Minimum and maximum current ( $I_{OUT}$  and  $V_{OUT}$ ) curves are given in Figure 9b for each of these devices to allow the designer to effectively use these I/O configurations in designing a COP420/421 system.

The SO, SK outputs can be configured as shown in a., b., or c. The D and G outputs can be configured as shown in a. or b. Note that when inputting data to the G ports, the G outputs should be set to "1." The L outputs can be configured as in d., e., f. or g.

An important point to remember if using configuration d. or f. with the L drivers is that even when the L drivers are disabled, the depletion load device will source a small amount of current (see Figure 9b, device 2); however, when the L lines are used as inputs, the disabled depletion device can *not* be relied on to source sufficient current to pull an input to logic "1".

**COP421**

If the COP420 is bonded as a 24-pin device, it becomes the COP421, illustrated in Figure 2, COP420/421 Connection Diagrams. Note that the COP421 does not contain the four general purpose IN inputs (IN<sub>3</sub> - IN<sub>0</sub>). Use of this option precludes, of course, use of the IN options, interrupt feature, and the MICROBUS™ option which uses IN<sub>1</sub> - IN<sub>3</sub>. All other options are available for the COP421.

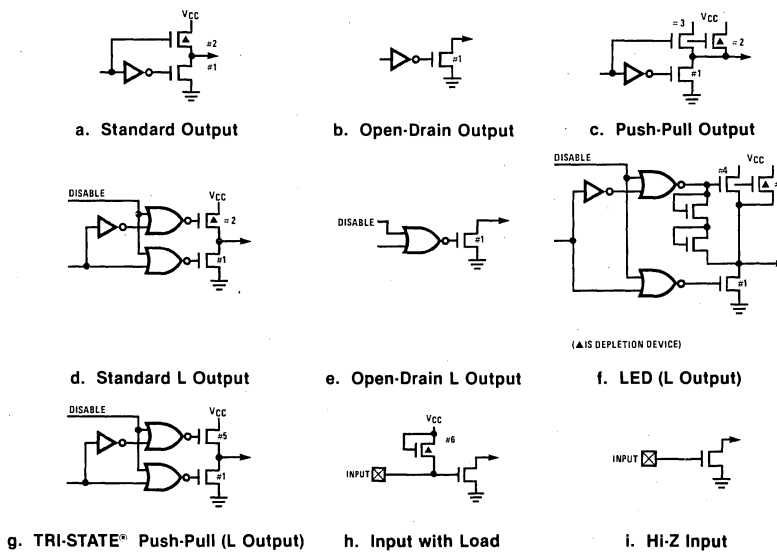


Figure 9a. Input/Output Configurations

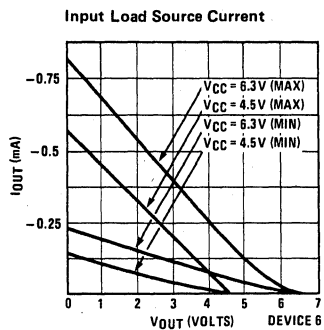
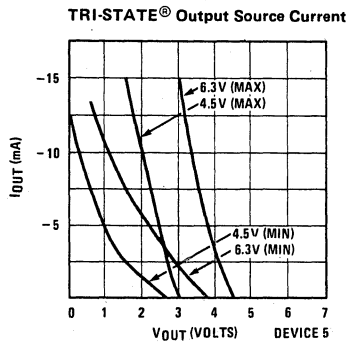
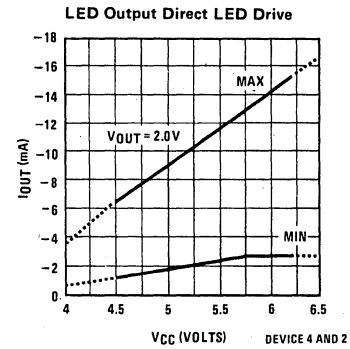
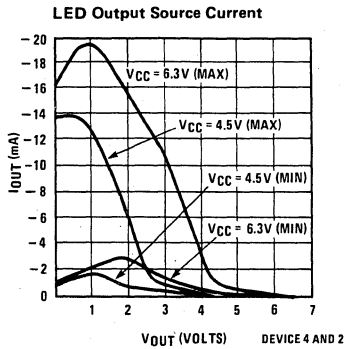
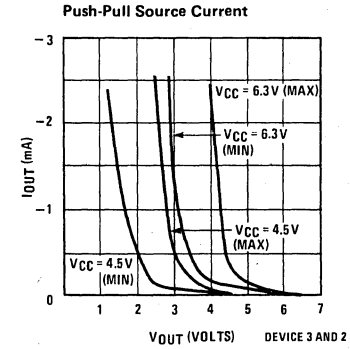
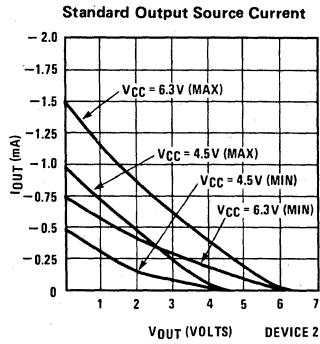
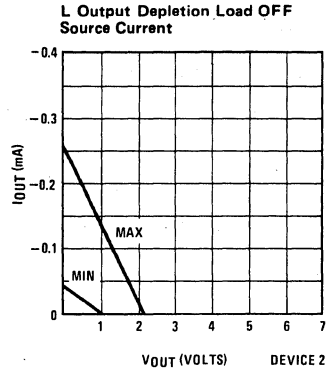
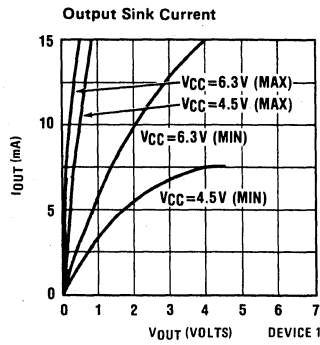


Figure 9b. COP420/COP421 Input/Output Characteristics

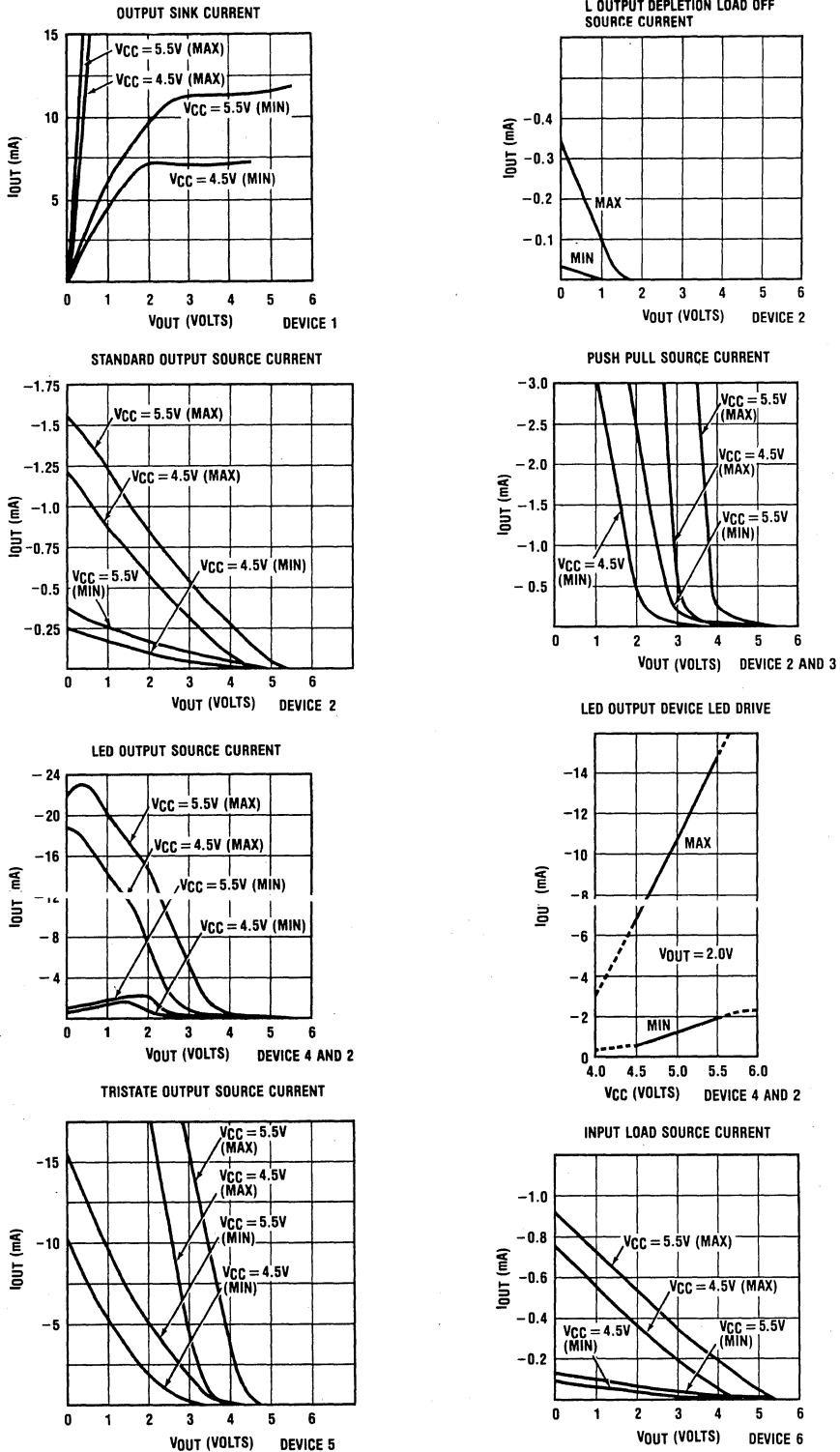


Figure 9c. COP320/COP321 Input/Output Characteristics

### COP420/COP421/COP422/COP320/COP321/COP322 Instruction Set

Table 1 is a symbol table providing internal architecture, instruction operand and operational symbols used in the instruction set table.

Table 2 provides the mnemonic, operand, machine code, data flow, skip conditions, and description associated with each instruction in the COP420/COP421/COP422 instruction set.

**Table 2. COP420/421/422/320/321/322 Instruction Set Table Symbols**

Symbol	Definition	Symbol	Definition
<b>INTERNAL ARCHITECTURE SYMBOLS</b>		<b>INSTRUCTION OPERAND SYMBOLS</b>	
A	4-bit Accumulator	d	4-bit Operand Field, 0–15 binary (RAM Digit Select)
B	6-bit RAM Address Register	r	2-bit Operand Field, 0–3 binary (RAM Register Select)
Br	Upper 2 bits of B (register address)	a	10-bit Operand Field, 0–1023 binary (ROM Address)
Bd	Lower 4 bits of B (digit address)	y	4-bit Operand Field, 0–15 binary (Immediate Data)
C	1-bit Carry Register	RAM(s)	Contents of RAM location addressed by s
D	4-bit Data Output Port	ROM(t)	Contents of ROM location addressed by t
EN	4-bit Enable Register	<b>OPERATIONAL SYMBOLS</b>	
G	4-bit Register to latch data for G I/O Port	+	Plus
IL	Two 1-bit latches associated with the IN <sub>3</sub> or IN <sub>0</sub> inputs	–	Minus
IN	4-bit Input Port	→	Replaces
L	8-bit TRI-STATE® I/O Port	↔	Is exchanged with
M	4-bit contents of RAM Memory pointed to by B Register	=	Is equal to
PC	10-bit ROM Address Register (program counter)	$\bar{A}$	The one's complement of A
Q	8-bit Register to latch data for L I/O Port	⊕	Exclusive-OR
SA	10-bit Subroutine Save Register A	:	Range of values
SB	10-bit Subroutine Save Register B		
SC	10 Subroutine Save Register A		
SIO	4-bit Shift Register and Counter		
SK	Logic-Controlled Clock Output		

**Table 2. COP420/421/422/320/321/322 Instruction Set**

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>ARITHMETIC INSTRUCTIONS</b>						
ASC		30	00110000	A + C + RAM(B) → A Carry → C	Carry	Add with Carry, Skip on Carry
ADD		31	00110001	A + RAM(B) → A	None	Add RAM to A
ADT		4A	01001010	A + 10 <sub>10</sub> → A	None	Add Ten to A
AISC	y	5-	0101 y	A + y → A	Carry	Add Immediate, Skip on Carry (y ≠ 0)
CASC		10	00010000	$\bar{A}$ + RAM(B) + C → A Carry → C	Carry	Complement and Add with Carry, Skip on Carry
CLRA		00	00000000	0 → A	None	Clear A
COMP		40	01000000	$\bar{A}$ → A	None	One's complement of A to A
NOP		44	01000100	None	None	No Operation
RC		32	00110010	"0" → C	None	Reset C
SC		22	00100010	"1" → C	None	Set C
XOR		02	00000010	A ⊕ RAM(B) → A	None	Exclusive-OR RAM with A

Table 2. COP420/421/422/320/321/322 Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>TRANSFER OF CONTROL INSTRUCTIONS</b>						
JID		FF	1111 1111	ROM(PC <sub>9:8</sub> , A,M) → PC <sub>7:0</sub>	None	Jump Indirect (Note 3)
JMP	a	6-	0110 00 a <sub>9:8</sub> a <sub>7:0</sub>	a → PC	None	Jump
JP	a	--	1  a <sub>6:0</sub> (pages 2,3 only) or 11  a <sub>5:0</sub> (all other pages)	a → PC <sub>6:0</sub> a → PC <sub>5:0</sub>	None	Jump within Page (Note 4)
JSRP	a	--	10  a <sub>5:0</sub>	PC+1 → SA → SB → SC 0010 → PC <sub>9:6</sub> a → PC <sub>5:0</sub>	None	Jump to Subroutine Page (Note 5)
JSR	a	6-	0110 10 a <sub>9:8</sub> a <sub>7:0</sub>	PC+1 → SA → SB → SC a → PC	None	Jump to Subroutine
RET		48	0100 1000	SC → SB → SA → PC	None	Return from Subroutine
RETSK		49	0100 1001	SC → SB → SA → PC	Always Skip on Return	Return from Subroutine then Skip
<b>MEMORY REFERENCE INSTRUCTIONS</b>						
CAMQ		33 3C	0011 0011 0011 1100	A → Q <sub>7:4</sub> RAM(B) → Q <sub>3:0</sub>	None	Copy A, RAM to Q
CQMA		33 2C	0011 0011 0010 1100	Q <sub>7:4</sub> → RAM(B) Q <sub>3:0</sub> → A	None	Copy Q to RAM, A
LD	r	-5	00  r  0101	RAM(B) → A	None	Load RAM into A, Exclusive OR B with r
LDD	r,d	23 --	0010 0011 00  r   d	RAM(r,d) → A	None	Load A with RAM pointed to directly by r,d
LQID		BF	1011 1111	ROM(PC <sub>9:8</sub> ,A,M) → Q SB → SC	None	Load Q Indirect (Note 3)
RMB	0 1 2 3	4C 45 42 43	0100 1100 0100 0101 0100 0010 0100 0011	0 → RAM(B) <sub>0</sub> 0 → RAM(B) <sub>1</sub> 0 → RAM(B) <sub>2</sub> 0 → RAM(B) <sub>3</sub>	None	Reset RAM Bit
SMB	0 1 2 3	4D 47 46 4B	0100 1101 0100 1101 0100 0110 0100 1011	1 → RAM(B) <sub>0</sub> 1 → RAM(B) <sub>1</sub> 1 → RAM(B) <sub>2</sub> 1 → RAM(B) <sub>3</sub>	None	Set RAM Bit

Table 2. COP420/421/422/320/321/322 Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>MEMORY REFERENCE INSTRUCTIONS (continued)</b>						
STII	y	7-	0111   y	y → RAM(B) Bd + 1 → Bd	None	Store Memory Immediate and Increment Bd
X	r	-6	00   r   0110	RAM(B) ↔ A Br ⊕ r → Br	None	Exchange RAM with A, Exclusive-OR Br with r
XAD	r,d	23 --	0010   0011 10   r   d	RAM(r,d) ↔ A	None	Exchange A with RAM pointed to directly by r,d
XDS	r	-7	00   r   0111	RAM(B) ↔ A Bd - 1 → Bd Br ⊕ r → Br	Bd decrements past 0	Exchange RAM with A and Decrement Bd, Exclusive-OR Br with r
XIS	r	-4	00   r   0100	RAM(B) ↔ A Bd + 1 → Bd Br ⊕ r → Br	Bd increments past 15	Exchange RAM with A and Increment Bd, Exclusive-OR Br with r
<b>REGISTER REFERENCE INSTRUCTIONS</b>						
CAB		50	0101   0000	A → Bd	None	Copy A to Bd
CBA		4E	0100   1110	Bd → A	None	Copy Bd to A
LBI	r,d	-- 33 --	00   r   (d-1) (d = 0, 9:15) or 0011   0011 10   r   d (any d)	r,d → B	Skip until not a LBI	Load B Immediate with r,d (Note 6)
LEI	y	33 6-	0011   0011 0110   y	y → EN	None	Load EN Immediate (Note 7)
XABR		12	0001   0010	A ↔ Br (0,0 → A <sub>3</sub> ,A <sub>2</sub> )	None	Exchange A with Br
<b>TEST INSTRUCTIONS</b>						
SKC		20	0010   0000		C = "1"	Skip if C is True
SKE		21	0010   0001		A = RAM(B)	Skip if A Equals RAM
SKGZ		33 21	0011   0011 0010   0001		G <sub>3:0</sub> = 0	Skip if G is Zero (all 4 bits)
SKGBZ		33	0011   0011	1st byte		Skip if G Bit is Zero
	0	01	0000   0001		G <sub>0</sub> = 0	
	1	11	0001   0001		G <sub>1</sub> = 0	
	2	03	0000   0011	2nd byte	G <sub>2</sub> = 0	
	3	13	0001   0011		G <sub>3</sub> = 0	
SKMBZ		0 1 2 3	0000   0001 0001   0001 0000   0011 0001   0011		RAM(B) <sub>0</sub> = 0 RAM(B) <sub>1</sub> = 0 RAM(B) <sub>2</sub> = 0 RAM(B) <sub>3</sub> = 0	Skip if RAM Bit is Zero
SKT		41	0100   0001		A time-base counter carry has occurred since last test	Skip on Timer (Note 3)

Table 2. COP420/421/422/320/321/322 Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
INPUT/OUTPUT INSTRUCTIONS						
ING		33	0011 0011	G → A	None	Input G Ports to A
		2A	0010 1010			
ININ		33	0011 0011	IN → A	None	Input IN Inputs to A (Note 2)
		28	0010 1000			
INIL		33	0011 0011	IL <sub>3</sub> , CKO, "0", IL <sub>0</sub> → A	None	Input IL Latches to A (Note 3)
		29	0010 1001			
INL		33	0011 0011	L <sub>7:4</sub> → RAM(B) L <sub>3:0</sub> → A	None	Input L Ports to RAM,A
		2E	0010 1110			
OBD		33	0011 0011	Bd → D	None	Output Bd to D Outputs
		3E	0011 1110			
OGI	y	33	0011 0011	y → G	None	Output to G Ports Immediate
		5-	0101 y			
OMG		33	0011 0011	RAM(B) → G	None	Output RAM to G Ports
		3A	0011 1010			
XAS		4F	0100 1111	A ↔ SIO, C → SKL	None	Exchange A with SIO (Note 3)

**Note 1:** All subscripts for alphabetical symbols indicate bit numbers unless explicitly defined (e.g., Br and Bd are explicitly defined). Bits are numbered 0 to N where 0 signifies the least significant bit (low-order, right-most bit). For example, A<sub>3</sub> indicates the most significant (left-most) bit of the 4-bit A register.

**Note 2:** The ININ instruction is not available on the COP421/COP321 and COP422/COP322 since these devices do not contain the IN inputs.

**Note 3:** For additional information on the operation of the XAS, JID, LQID, INIL, and SKT instructions, see below.

**Note 4:** The JP instruction allows a jump, while in subroutine pages 2 or 3, to any ROM location within the two-page boundary of pages 2 or 3. The JP instruction, otherwise, permits a jump to a ROM location within the current 64-word page. JP may not jump to the last word of a page.

**Note 5:** A JSRP transfers program control to subroutine page 2 (0010 is loaded into the upper 4 bits of P). A JSRP may not be used when in pages 2 or 3. JSRP may not jump to the last word in page 2.

**Note 6:** LBI is a single-byte instruction if d = 0, 9, 10, 11, 12, 13, 14, or 15. The machine code for the lower 4 bits equals the binary value of the "d" data minus 1, e.g., to load the lower four bits of B (Bd) with the value 9 (1001<sub>2</sub>), the lower 4 bits of the LBI instruction equal 8 (1000<sub>2</sub>). To load 0, the lower 4 bits of the LBI instruction should equal 15 (1111<sub>2</sub>).

**Note 7:** Machine code for operand field y for LEI instruction should equal the binary value to be latched into EN, where a "1" or "0" in each bit of EN corresponds with the selection or deselection of a particular function associated with each bit. (See Functional Description, EN Register.)



The following information is provided to assist the user in understanding the operation of several unique instructions and to provide notes useful to programmers in writing COP420/421 programs.

### XAS Instruction

XAS (Exchange A with SIO) exchanges the 4-bit contents of the accumulator with the 4-bit contents of the SIO register. The contents of SIO will contain serial-in/serial-out shift register or binary counter data, depending on the value of the EN register. An XAS instruction will also affect the SK output. (See Functional Description, EN Register, above.) If SIO is selected as a shift register, an XAS instruction must be performed once every 4 instruction cycles to effect a continuous data stream.

### JID Instruction

JID (Jump Indirect) is an indirect addressing instruction, transferring program control to a new ROM location pointed to indirectly by A and M. It loads the lower 8 bits of the ROM address register PC with the contents of ROM addressed by the 10-bit word, PC<sub>9:8</sub>, A, M. PC<sub>9</sub> and PC<sub>8</sub> are not affected by this instruction.

Note that JID requires 2 instruction cycles to execute.

### INIL Instruction

INIL (Input IL Latches to A) inputs 2 latches, IL<sub>3</sub> and IL<sub>0</sub> (see figure 10) and CKO into A. The IL<sub>3</sub> and IL<sub>0</sub> latches are set if a low-going pulse ("1" to "0") has occurred on the IN<sub>3</sub> and IN<sub>0</sub> inputs since the last INIL instruction, provided the input pulse stays low for at least two instruction times. Execution of an INIL inputs IL<sub>3</sub> and IL<sub>0</sub> into A3 and A0 respectively, and resets these latches to allow them to respond to subsequent low-going pulses on the IN<sub>3</sub> and IN<sub>0</sub> lines. If CKO is mask programmed as a general purpose input, an INIL will input the state of CKO into A2. If CKO has not been so programmed, a "1" will be placed in A2. A "0" is always placed in A1 upon the execution of an INIL. The general purpose inputs IN<sub>3</sub>-IN<sub>0</sub> are input to A upon execution of an ININ instruction. (See table 2, ININ instruction.) INIL is useful in recognizing pulses of short duration or pulses which occur too often to be read conveniently by an ININ instruction.

Note: IL latches are not cleared on reset.

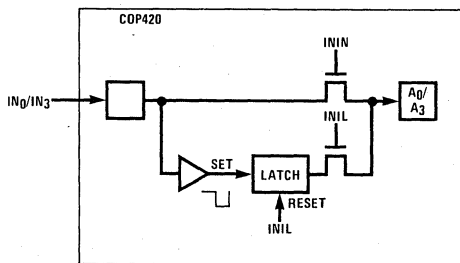


Figure 10.

### LQID Instruction

LQID (Load Q Indirect) loads the 8-bit Q register with the contents of ROM pointed to by the 10-bit word PC<sub>9</sub>, PC<sub>8</sub>, A, M. LQID can be used for table lookup or code conversion such as BCD to seven-segment. The LQID instruction "pushes" the stack (PC + 1 → SA → SB → SC) and replaces the least significant 8 bits of PC as follows: A → PC<sub>7:4</sub>; RAM(B) → PC<sub>3:0</sub>, leaving PC<sub>9</sub> and PC<sub>8</sub> unchanged. The ROM data pointed to by the new address is fetched and loaded into the Q latches. Next, the stack is "popped" (SC → SB → SA → PC), restoring the saved value of PC to continue sequential program execution. Since LQID pushes SB → SC, the previous contents of SC are lost. Also, when LQID pops the stack, the previously pushed contents of SB are left in SC. The net result is that the contents of SB are placed in SC (SB → SC). Note that LQID takes two instruction cycle times to execute.

### SKT Instruction

The SKT (Skip On Timer) instruction tests the state of an internal 10-bit time-base counter. This counter divides the instruction cycle clock frequency by 1024 and provides a latched indication of counter overflow. The SKT instruction tests this latch, executing the next program instruction if the latch is not set. If the latch has been set since the previous test, the next program instruction is skipped and the latch is reset. The features associated with this instruction, therefore, allow the COP420/421 to generate its own time-base for real-time processing rather than relying on an external input signal.

For example, using a 2.097 MHz crystal as the time-base to the clock generator, the instruction cycle clock frequency will be 131 kHz (crystal frequency ÷ 16) and the binary counter output pulse frequency will be 128 Hz. For time-of-day or similar real-time processing, the SKT instruction can call a routine which increments a "seconds" counter every 128 ticks.

### Instruction Set Notes

- The first word of a COP420/421 program (ROM address 0) must be a CLRA (Clear A) instruction.
- Although skipped instructions are not executed, one instruction cycle time is devoted to skipping each byte of the skipped instruction. Thus all program paths take the same number of cycle times whether instructions are skipped or executed except JID and LQID. LQID and JID take two cycle times if executed and one if skipped.
- The ROM is organized into 16 pages of 64 words each. The Program Counter is an 10-bit binary counter, and will count through page boundaries. If a JP, JSRP, JID or LQID instruction is located in the last word of a page, the instruction operates as if it were in the next page. For example: a JP located in the last word of a page will jump to a location in the next page. Also, a LQID or JID located in the last word of page 3, 7, 11 or 15 will access data in the next group of four pages.

## Option List

The COP420/421/422 mask-programmable options are assigned numbers which correspond with the COP420 pins.

The following is a list of COP420 options. When specifying a COP421 or COP422 chip, Options 9, 10, 19, 20 and 29 must all be set to zero. When specifying a COP422 chip, Options 21, 22, 27 and 28 must also be zero, and Option 2 must not be a 1. The options are programmed at the same time as the ROM pattern to provide the user with the hardware flexibility to interface to various I/O components using little or no external circuitry.

Option 1 = 0: Ground Pin — no options available

Option 2: CKO Pin

- = 0: clock generator output to crystal (0 not available if option 3 = 4 or 5)
- = 1: pin is RAM power supply ( $V_R$ ) input (Not available on COP422/COP322)
- = 2: general purpose input with load device
- = 3: multi-COP SYNC input
- = 4: general purpose Hi Z input

Option 3: CKI Input

- = 0: crystal input divided by 16
- = 1: crystal input divided by 8
- = 2: TTL external clock input divided by 16
- = 3: TTL external clock input divided by 8
- = 4: single-pin RC controlled oscillator (+4)
- = 5: Schmitt trigger clock input (+4)

Option 4: RESET Pin

- = 0: Load devices to  $V_{CC}$
- = 1: Hi-Z input

Option 5: L<sub>7</sub> Driver

- = 0: Standard output (figure 9D)
- = 1: Open-Drain output (E)
- = 2: LED direct drive output (F)
- = 3: TRI-STATE push-pull output (G)

Option 6: L<sub>6</sub> Driver

same as Option 5

Option 7: L<sub>5</sub> Driver

same as Option 5

Option 8: L<sub>4</sub> Driver

same as Option 5

Option 9: IN<sub>1</sub> Input

- = 0: load device to  $V_{CC}$  (H)
- = 1: Hi-Z input (I)

Option 10: IN<sub>2</sub> Input

same as Option 9

Option 11 = 0:  $V_{CC}$  Pin — no options available

Option 12: L<sub>3</sub> Driver

same as Option 5

Option 13: L<sub>2</sub> Driver

same as Option 5

Option 14: L<sub>1</sub> Driver

same as Option 5

Option 15: L<sub>0</sub> Driver

same as Option 5

Option 16: SI Input

same as Option 9

Option 17: SO Driver

- = 0: standard output (A)
- = 1: open-drain output (B)
- = 2: push-pull output (C)

Option 18: SK Driver

same as Option 17

Option 19: IN<sub>0</sub> Input

same as Option 9

Option 20: IN<sub>3</sub> Input

same as Option 9

Option 21: G<sub>0</sub> I/O Port

- = 0: Standard output (A)
- = 1: Open-Drain output (B)

Option 22: G<sub>1</sub> I/O Port

same as Option 21

Option 23: G<sub>2</sub> I/O Port

same as Option 21

Option 24: G<sub>3</sub> I/O Port

same as Option 21

Option 25: D<sub>3</sub> Output

- = 0: Standard output (A)
- = 1: Open-Drain output (B)

Option 26: D<sub>2</sub> Output

same as Option 25

Option 27: D<sub>1</sub> Output

same as Option 25

Option 28: D<sub>0</sub> Output

same as Option 25

Option 29: COP Function

- = 0: normal operation
- = 1: MICROBUS™ option

Option 30: COP Bonding

- = 0: COP420 (28-pin device)
- = 1: COP421 (24-pin device)
- = 2: 28- and 24-pin device
- = 3: COP422 (20-pin device)
- = 4: 28- and 20-pin device
- = 5: 24- and 20-pin device
- = 6: 28-, 24- and 20-pin device

Option 31: IN Input Levels

- = 0: normal input levels
- = 1: Higher voltage input levels ("0" = 1.2V, "1" = 3.6V)

Option 32: G Input Levels

same as Option 31

Option 33: L Input Levels

same as Option 31

Option 34: CKO Input Levels

same as Option 31

Option 35: SI Input Levels

same as Option 31

### TEST MODE (Non-Standard Operation)

The SO output has been configured to provide for standard test procedures for the custom-programmed COP420. With SO forced to logic "1," two test modes are provided, depending upon the value of SI:

- a. RAM and Internal Logic Test Mode (SI = 1)
- b. ROM Test Mode (SI = 0)

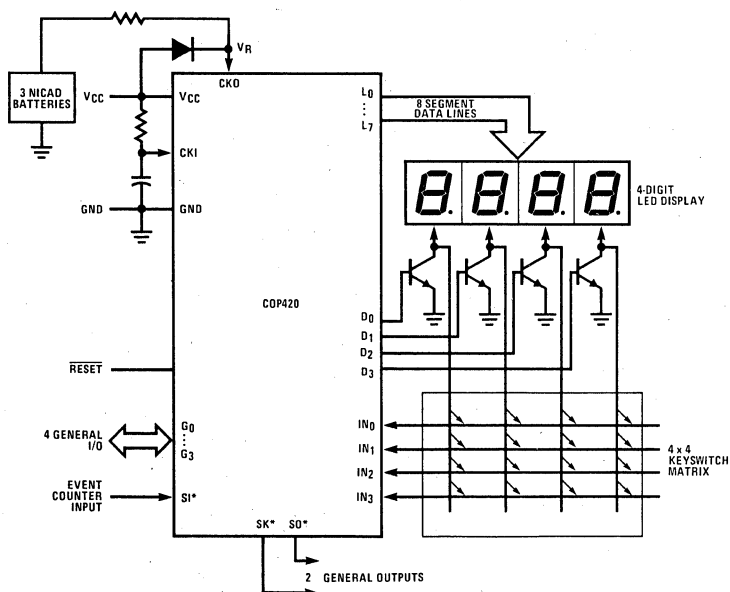
These special test modes should not be employed by the user; they are intended for manufacturing test only.

### APPLICATION #1: COP420 General Controller

Figure 8 shows an interconnect diagram for a COP420 used as a general controller. Operation of the system is as follows:

1. The L<sub>7</sub>-L<sub>0</sub> outputs are configured as LED Direct Drive outputs, allowing direct connection to the segments of the display.

2. The D<sub>3</sub>-D<sub>0</sub> outputs drive the digits of the multiplexed display directly and scan the columns of the 4 × 4 keyboard matrix.
3. The IN<sub>3</sub>-IN<sub>0</sub> inputs are used to input the 4 rows of the keyboard matrix. Reading the IN lines in conjunction with the current value of the D outputs allows detection, debouncing, and decoding of any one of the 16 keyswitches.
4. CKI is configured as a single-pin oscillator input allowing system timing to be controlled by a single-pin RC network. CKO is therefore available for use as a V<sub>R</sub> RAM power supply pin. RAM data integrity is thereby assured when the main power supply is shut down (see RAM Keep-Alive Option description).
5. SI is selected as the input to a binary counter input. With SIO used as a binary counter, SO and SK can be used as general purpose outputs.
6. The 4 bidirectional G I/O ports (G<sub>3</sub>-G<sub>0</sub>) are available for use as required by the user's application.



\*SI, SO and SK may also be used for serial I/O

Figure 11. COP420 Keyboard/Display Interface

## APPLICATION #2: Musical Organ and Music Box

**Play Mode:** Twenty-five musical keys and 25 LEDs are provided to denote F to F with half notes in between. All the keys and LEDs are directly detected and driven by the microprocessor. Depression of the key will give the corresponding musical note and light up the corresponding LED.

**Clear:** Memory is provided to store a played tune. Depression of the CLEAR key erases the memory and the microprocessor is ready to store new musical notes. A maximum of 28 notes can be stored where each note can be of one to eight musical beats. (Two bytes of memory are required to store one musical note. Any note longer than eight musical beats will require additional memory space for storage.)

**Playback:** Depression of this button will playback the tune stored in the memory since last "clear."

**Preprogrammed Tunes:** There are ten preprogrammed tunes (each has an average of 55 notes) masked in the chip. Any tune can be recalled by depressing the "Tune Button" followed by the corresponding "Sharp Key."

**Learn Mode:** This mode is for the player to learn the ten preprogrammed tunes. By pressing the "Learn Button"

followed by the corresponding "Sharp Key," the LEDs will be lighted up one by one to indicate the notes of the selected tune. The LED will remain "on" until the player presses the correct musical key; the LED for the next note will then be lighted up.

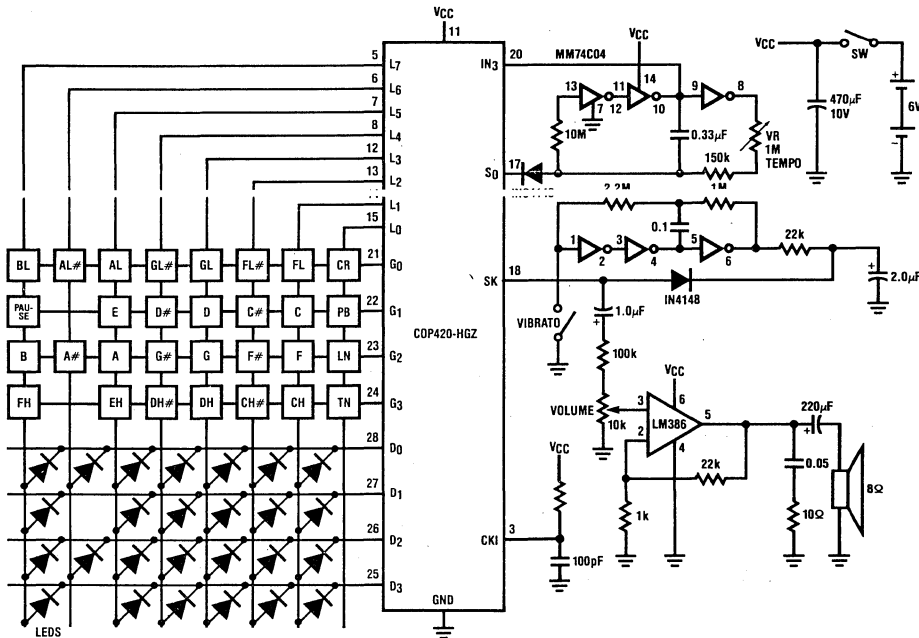
**Pause:** In addition to the 25 musical keys, there is a special pause key. The depression of this key generates a blank note to the memory.

**Note:** In the Learn Mode when playing "Oh Susanna," the pause key must be used.

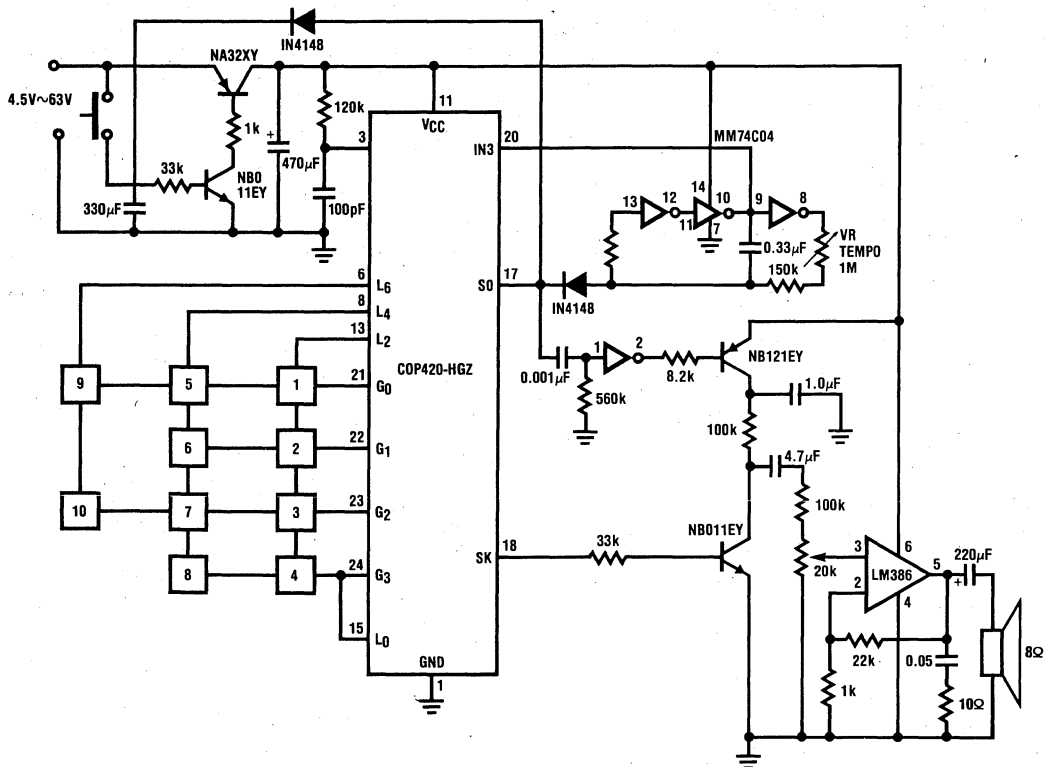
**Tempo:** This is a control input to the musical beat time oscillator for varying the speed of the musical tunes.

**Vibrato:** This is a switch control to vary the frequency vibration of the note.

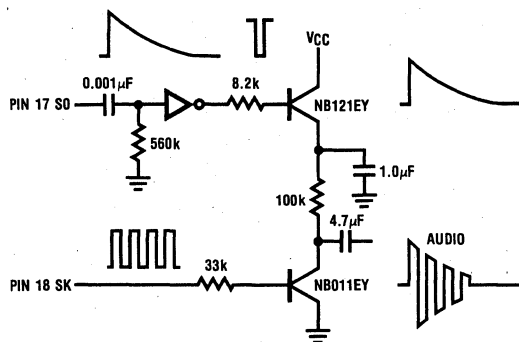
**Tunes Listing:** The following is a listing of the ten preprogrammed tunes: 1) Jingle Bells, 2) Twinkle, Twinkle Little Star, 3) Happy Birthday, 4) Yankee Doodle, 5) Silent Night, 6) This Old Man, 7) London Bridge Is Falling Down, 8) Auld Lang Syne, 9) Oh Susanna, 10) Clementine.



Circuit Diagram of COP420 Musical Organ

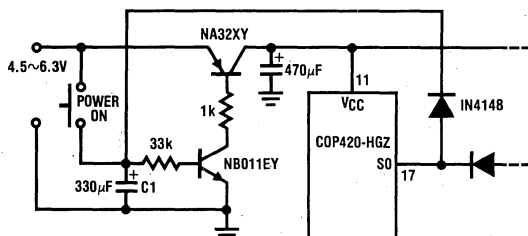


Music Box Application with Direct Key Access



This additional circuit provides tinkling effect for the musical note.

Bell Sound Circuit



This circuit automatically turns off the musical organ if none of the keys are pressed with in approximately 30 seconds.

Auto Power Shut-Off Circuit

## COP420C/COP421C and COP320C/COP321C Single-Chip CMOS Microcontrollers

### General Description

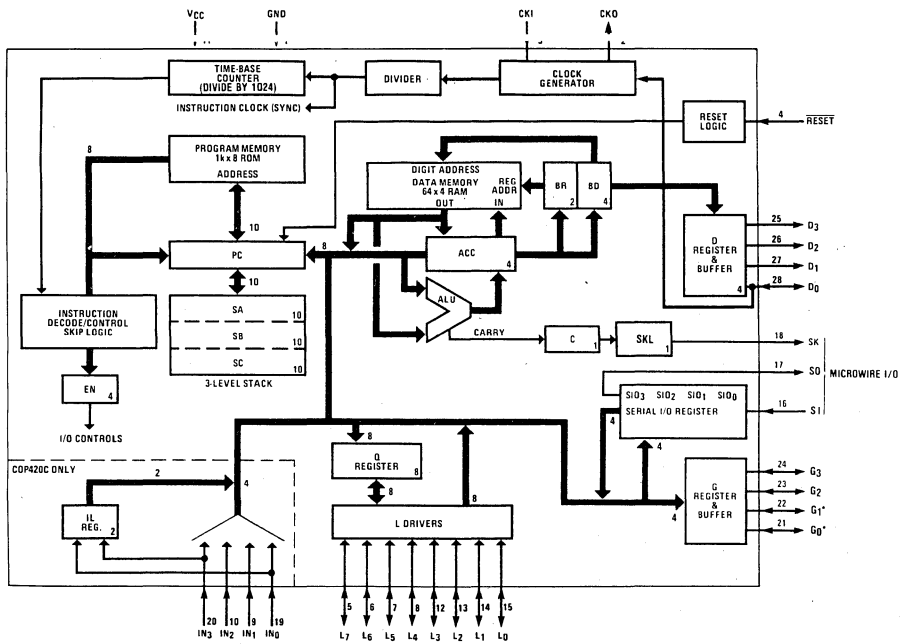
The COP420C, COP421C, COP320C, and COP321C Single-Chip CMOS Microcontrollers are members of the COPS™ family, fabricated using complementary MOS technology. They are complete microcomputers containing all system timing, internal logic, ROM, RAM and I/O necessary to implement dedicated control functions in a variety of applications. Features include single supply operation, a variety of output configuration options, with an instruction set, internal architecture and I/O scheme designed to facilitate keyboard input, display output and BCD and binary data manipulation. The COP421C is identical to the COP420C, except with 19 I/O lines instead of 23. They are an appropriate choice for use in numerous human interface control environments. Standard test procedures and reliable high-density fabrication techniques provide the medium to large volume customers with a customized Control Oriented Processor at a low end-product cost.

The COP320C is the extended temperature range version of the COP420C (likewise the COP321C is the extended temperature range version of the COP421C). The COP320C/321C are exact functional equivalents of the COP420C/421C.

### Features

- Lowest power dissipation (50 $\mu$ W typical)
- Power saving "Idle" state
- Powerful instruction set
- 1k  $\times$  8 ROM, 64  $\times$  4 RAM, 23 I/O lines (COP420C)
- True vectored interrupt, plus restart
- Three-level subroutine stack
- 15 $\mu$ s instruction time, plus software selectable oscillators
- Single supply operation (2.4-5.5V)
- Internal time-base counter for real-time processing
- MICROWIRE™ compatible serial I/O
- General purpose and TRI-STATE® outputs
- LSTTL/CMOS compatible
- MICROBUS™ compatible
- Software/hardware compatible with other members of COP400 family
- Extended temperature range device COP320C/COP321C (-40°C to +85°C)

### COP420C/421C and COP320C/321C Block Diagram



## COP420C/COP421C and COP320C/COP321C

## Absolute Maximum Ratings

Voltage at Any Pin	-0.3V to $V_{CC} + 0.3V$	Package Power Dissipation	700mW at 25°C
Operating Temperature Range			300mW at 70°C
COP420C/COP421C	0°C to 70°C		150mW at 85°C
COP320C/COP321C	-40°C to +85°C	Total Sink Current	40mA
Storage Temperature Range	-65°C to +150°C	Total Source Current	40mA
Lead Temperature (Soldering, 10 seconds)	300°C		

Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.

## DC Electrical Characteristics

COP420C/421C:  $0^{\circ}\text{C} \leq T_A \leq 70^{\circ}\text{C}$ ,  $2.4\text{V} \leq V_{CC} \leq 5.5\text{V}$  unless otherwise noted.

COP320C/321C:  $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $3.0\text{V} \leq V_{CC} \leq 5.3\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Operation Voltage	COP420C/421C COP320C/321C	2.4 3.0	5.5 5.3	V V
Power Supply Ripple	peak to peak (Note 1)		$0.1V_{CC}$	V
Supply Current	$V_{CC} = 2.4\text{V}$ , $f_{IN} = 32\text{kHz}$ ( $\pm 8$ mode) $V_{CC} = 5.0\text{V}$ , $f_{IN} = 32\text{kHz}$ ( $\pm 8$ mode) $V_{CC} = 5.0\text{V}$ , $f_{IN} = \text{Max.}$ ( $\pm 8$ mode) $V_{CC} = 5.0\text{V}$ , $f_{IN} = \text{Max.}$ ( $\pm 16$ mode)		35 100 800 1200	$\mu\text{A}$ $\mu\text{A}$ $\mu\text{A}$ $\mu\text{A}$
Idle State Current	$V_{CC} = 2.4\text{V}$ , $f_{IN} = 32\text{kHz}$ $V_{CC} = 5.0\text{V}$ , $f_{IN} = \text{Max.}$		15 250	$\mu\text{A}$ $\mu\text{A}$
Input Voltage Levels				
Schmitt Trigger Inputs				
RESET, DO (as clock)				
Logic High		$0.9V_{CC}$		V
Logic Low			$0.1V_{CC}$	V
All Other Inputs				
Logic High		$0.6V_{CC}$		V
Logic Low			$0.25V_{CC}$	V
Output Voltage levels				
Standard Outputs				
LSTTL Operation				
Logic High	$V_{CC} = 5V \pm 5\%$	2.7		V
Logic Low	$I_{OH} = -100\mu\text{A}$ $I_{OL} = 0.4\text{mA}$		0.4	V
CMOS Operation				
Logic High	$V_{CC} > 3V$ $I_{OH} = -10\mu\text{A}$	$V_{CC} - 0.2$		V
Logic Low	$I_{OL} = 10\mu\text{A}$		0.2	V
Output Current Levels				
Sink Current	$V_{OUT} = V_{CC}$			
CKO	$V_{CC} = 5V$	100		$\mu\text{A}$
All Others	$V_{CC} = 5V$	1.2		mA
All Others	$V_{CC} = \text{Min.}$	0.2		mA
Source Current	$V_{OUT} = 0V$			
CKO	$V_{CC} = 5V$	-100		$\mu\text{A}$
All Others	$V_{CC} = 5V$	-0.2		mA
All Others	$V_{CC} = \text{Min.}$	-0.1		mA

**DC Electrical Characteristics** (Cont'd)COP420C/421C:  $0^{\circ}\text{C} \leq T_A \leq 70^{\circ}\text{C}$ ,  $2.4\text{V} \leq V_{CC} \leq 5.5\text{V}$  unless otherwise noted.COP320C/321C:  $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $3.0\text{V} \leq V_{CC} \leq 5.3\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Allowable Sink Current				
Per Pin (SO, SK, CKO)			2	mA
Per Pin (All Others)			8	mA
Per Port (L)			16	mA
Per Port (D, G)			8	mA
Allowable Source Current				
Per Pin			-5	mA
Input Load Source Current	$V_{CC} = 5\text{V}$ , $V_{IN} = 0$	-25	-300	$\mu\text{A}$
	$V_{CC} = \text{Min.}$ , $V_{IN} = 0$	-6	-75	$\mu\text{A}$
Hi-Z Input Leakage	$V_{CC} = 5\text{V}$ (COP420C/421C)	-1	+1	$\mu\text{A}$
	$V_{CC} = 5\text{V}$ (COP320C/321C)	-2	+2	$\mu\text{A}$
TRI-STATE® or Open Drain Leakage Current	(COP420C/421C)	-2.5	+2.5	$\mu\text{A}$
	(COP320C/321C)	-5	+5	$\mu\text{A}$



## AC Electrical Characteristics

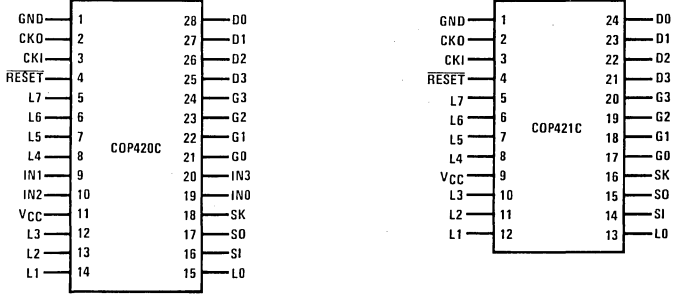
COP420C/COP421C:  $0^{\circ}\text{C} \leq T_A \leq 70^{\circ}\text{C}$ ,  $2.4\text{V} \leq V_A \leq 5.5\text{V}$  unless otherwise noted.

COP320C/COP321C:  $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $3.0\text{V} \leq V_{CC} \leq 5.3\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Instruction Cycle Time COP420C/421C	$V_{CC} \geq 4.5\text{V}$	15	245	$\mu\text{s}$
	$V_{CC} \geq 2.4\text{V}$	50	245	$\mu\text{s}$
Operating CKI Frequency COP420C/421C	$\div 8$ mode	32	500	kHz
	$\div 16$ mode	64	1000	kHz
	$\div 32$ mode	128	2097	kHz
	Dual Clk or IT		500	kHz
	$\div 8$ mode	32	160	kHz
	$\div 16$ mode	64	320	kHz
	$\div 32$ mode	128	640	kHz
	Dual Clk or IT		160	kHz
Instruction Cycle Time COP320C/321C	$V_{CC} > 4.5\text{V}$	20	125	$\mu\text{s}$
	$V_{CC} > 3.0\text{V}$	50	125	$\mu\text{s}$
Operating CKI Frequency COP320C/321C	$\div 8$ mode	64	400	kHz
	$\div 16$ mode	128	800	kHz
	$\div 32$ mode	256	1600	kHz
	Dual Clk or IT		400	kHz
	$\div 8$ mode	64	160	kHz
	$\div 16$ mode	128	320	kHz
	$\div 32$ mode	256	640	kHz
	Dual Clk or IT		160	kHz
CKI Duty Cycle		30	50	%
Inputs:				
$t_{\text{SETUP}}$		2.0		$\mu\text{s}$
$t_{\text{HOLD}}$		0.6		$\mu\text{s}$
Output Propagation Delay $t_{\text{pd1}}$ $t_{\text{pd0}}$	Test Conditions: $V_{CC} > 4.5\text{V}$ , $R_L = 5\text{k}\Omega$ , $C_L = 50\text{pF}$ , $V_{\text{OUT}} = 1.5\text{V}$		6	$\mu\text{s}$
			6	$\mu\text{s}$
MICROBUS™ Timing	$C_L = 50\text{pF}$ , $V_{CC} = 5\text{V} \pm 5\%$			
Read Operation (Figure 4)				
Chip Select Stable before $\overline{\text{RD}}$ — $t_{\text{CSR}}$		65		ns
Chip Select Hold Time for $\overline{\text{RD}}$ — $t_{\text{RCS}}$		20		ns
$\overline{\text{RD}}$ Pulse Width— $t_{\text{RR}}$		400		ns
Data Delay from $\overline{\text{RD}}$ — $t_{\text{RD}}$			375	ns
$\overline{\text{RD}}$ to Data Floating— $t_{\text{DF}}$			250	ns
Write Operation (Figure 5)				
Chip Select Stable before $\overline{\text{WR}}$ — $t_{\text{CSW}}$		65		ns
Chip Select Hold Time for $\overline{\text{WR}}$ — $t_{\text{WCS}}$		20		ns
$\overline{\text{WR}}$ Pulse Width— $t_{\text{WW}}$		400		ns
Data Set-Up Time for $\overline{\text{WR}}$ — $t_{\text{DW}}$		320		ns
Data Hold Time for $\overline{\text{WR}}$ — $t_{\text{WD}}$		100		ns
INTR Transition Time from $\overline{\text{WR}}$ — $t_{\text{WI}}$			700	ns

**Note 1:** Voltage change must be less than 0.5 volts in a 1 ms period.

**Note 2:** Supply current is measured on the  $V_{CC}$  pin with a square wave clock, all inputs at  $V_{CC}$ ,  $\text{SO} = 1$ ,  $L_0 - L_7 = 0$  and outputs open. See COP Brief #14 for further information.



Order Number COP420C/N, COP320C/N  
NS Package N28A

Order Number COP421C/N  
NS Package N24A

Figure 2. Connection Diagrams

Pin	Description	Pin	Description
L7-L0	8 bidirectional I/O ports with TRI-STATE®	SK	Logic-controlled clock
G3-G0	4 bidirectional I/O ports	CKI	System oscillator input
D3-D1	3 general purpose outputs	CKO	System oscillator output (or general purpose input)
D0	General purpose output or oscillator input	RESET	System reset input
IN3-IN0	4 general purpose inputs (COP420C only)	VCC	Power supply
SI	Serial input	GND	Ground
SO	Serial output		

2

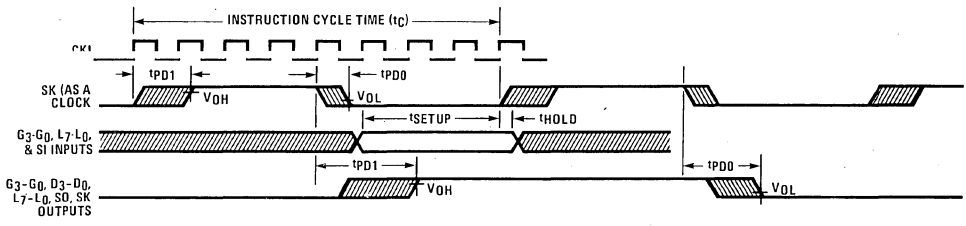


Figure 3. Input/Output Timing Diagrams (divide by 8 mode)

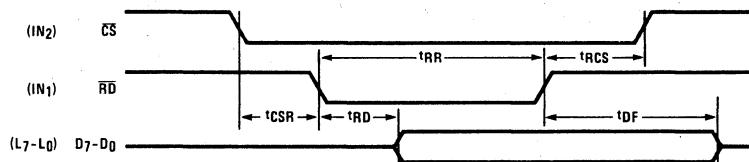


Figure 4. MICROBUST™ Read Operation Timing

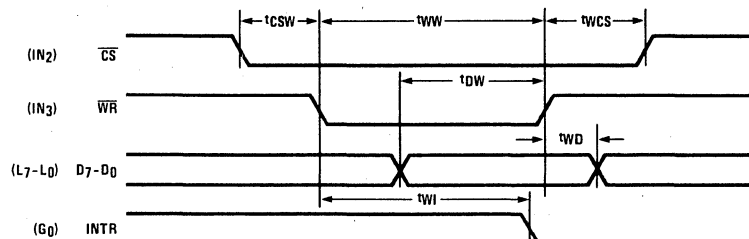


Figure 5. MICROBUST™ Write Operation Timing

## Functional Description

For ease of reading this description, only COP420C and/or COP421C are referenced; however, all such references apply equally to COP320C and/or COP321C, respectively.

A block diagram of the COP420C is given in Figure 1. Data paths are illustrated in simplified form to depict how the various logic elements communicate with each other in implementing the instruction set of the device. Positive logic is used. When a bit is set, it is a logic "1". When a bit is reset, it is a logic "0".

### Program Memory

Program Memory consists of a 1,024-byte ROM. As can be seen by an examination of the COP420C/421C instruction set, these words may be program instructions, program data or ROM addressing data. Because of the special characteristics associated with the JP, JSRP, JID and LQID instructions, ROM must often be thought of as being organized into 16 pages of 64 words each.

ROM addressing is accomplished by a 10-bit **PC register**. Its binary value selects one of the 1,024 8-bit words contained in ROM. A new address is loaded into the PC register during each instruction cycle. Unless the instruction is a transfer of control instruction, the PC register is loaded with the next sequential **10-bit binary count** value. Three levels of subroutine nesting are implemented by the 10-bit binary subroutine save registers, SA, SB and SC, providing a last-in, first-out (LIFO) hardware subroutine stack.

ROM instruction words are fetched, decoded and executed by the Instruction Decode, Control and Skip Logic circuitry.

### Data Memory

Data Memory consists of a 256-bit RAM, organized as 4 data registers of 16 4-bit digits. RAM addressing is implemented by a 6-bit **B-register** whose upper 2 bits (Br) select 1 of 4 data registers and lower 4 bits (Bd)

select 1 of 16 4-bit digits in the selected data register. While the 4-bit contents of the selected RAM digit (M) is usually loaded into or from, or exchanged with, the A register (accumulator), it may also be loaded into or from the Q latches or loaded from the L ports. RAM addressing may also be performed directly by the LDD and XAD instructions based upon the 6-bit contents of the operand field of these instructions. The Bd register also serves as a source register for 4-bit data sent directly to the D outputs.

### Internal Logic

The 4-bit **A register** (accumulator) is the source and destination register for most I/O, arithmetic, logic and data memory access operations. It can also be used to load the Br and Bd portions of the B register, to load and input 4 bits of the 8-bit Q latch data, to input 4 bits of the 8-bit L I/O port data and to perform data exchanges with the SIO register.

A **4-bit adder** performs the arithmetic and logic functions of the COP420C/421C, storing its results in A. It also outputs a carry bit to the 1-bit **C register**, most often employed to indicate arithmetic overflow. The C register, in conjunction with the XAS instruction and the EN register, also serves to control the SK output, C can be outputted directly to SK or can enable the SK to be a sync clock each instruction cycle time. (See XAS instruction and EN register description, below.)

Four **general-purpose inputs**, IN<sub>3</sub>-IN<sub>0</sub>, are provided; IN<sub>1</sub>, IN<sub>2</sub> and IN<sub>3</sub> may be selected, by a mask-programmable option, as Read Strobe, Chip Select and Write Strobe inputs, respectively, for use in MICROBUST™ applications.

The **D register** provides 4 general purpose outputs and is used as the destination register for the 4-bit contents of Bd. In the dual clock mode, D-register bit 0 controls the clock selection (see dual oscillator below).

The **G register** contents are outputs to 4 general-purpose bidirectional I/O ports.  $G_0$  may be mask-programmed as an output for MICROBUS™ applications.

The **Q register** is an internal, latched, 8-bit register, used to hold data loaded to or from M and A, as well as 8-bit data from ROM. Its contents are output to the L I/O ports when the L drivers are enabled under program control (see LEI instruction). With the MICROBUS™ option selected, Q can also be loaded with the 8-bit contents of the L I/O ports upon the occurrence of a write strobe from the host CPU.

The **8 L drivers**, when enabled, output the contents of latched Q data to the L I/O ports. Also, the contents of L may be read directly into A and M. As explained above, the MICROBUS™ option allows L I/O port data to be latched into the Q register.

The **SIO register** functions as a 4-bit serial-in/serial-out serial shift register shifting left each instruction cycle time. The data present at SI goes into the least significant bit of SIO. SO can be enabled to output the most significant bit of SIO each cycle time (see 4 below). The SK output becomes a logic-controlled clock. The SIO contents can be exchanged with A, allowing it to input or output a continuous serial data stream. SIO may also be used to provide additional parallel I/O by connecting SO to external serial-in/parallel-out shift registers.

The XAS instruction copies C into the **SKL Latch**. SK outputs SKL ANDed with the internal instruction cycle clock.

The **EN Register** is an internal 4-bit register loaded under program control by the LEI instruction. The state of each bit of this register selects or deselects the particular feature associated with each bit of the EN register ( $EN_3$ - $EN_0$ ).

1.  $EN_0$  controls the SO and SK outputs. With  $EN_0$  reset, SK is a logic-controlled clock and SO is serial data out. With  $EN_0$  set, SO and SK become general-purpose outputs.
2. With  $EN_1$  set the  $IN_1$  input is enabled as an interrupt input. Immediately following an interrupt,  $EN_1$  is reset to disable further interrupts.
3. With  $EN_2$  set, the L drivers are enabled to output the data in Q to the L I/O ports. Resetting  $EN_2$  disables the L drivers, placing the L I/O ports in a high-impedance input state. If the MICROBUS™ option is being used,  $EN_2$  does not affect the L drivers.
4.  $EN_3$ , in conjunction with  $EN_0$ , affects the SO output. If  $EN_0 = 0$ , setting  $EN_3$  enables SO as the output of the SIO shift register, outputting serial shifted data each instruction time. Resetting  $EN_3$  disables SO as the shift register output: data continues to be shifted through SIO and can be exchanged with A via an XAS instruction, but SO remains reset to "0". If  $EN_0 = 1$ , SO will output the value of  $EN_3$ . The table below provides a summary of  $EN_3$  and  $EN_0$ .

WARNING: If  $EN_0$  is set, do *NOT* use the contents of SIO.

Enable Register Modes — Bits  $EN_3$  and  $EN_0$

$EN_3$	$EN_0$	SIO	SI	SO	SK
0	0	Shift Register	Input to Shift Register	0	If SKL = 1, SK = Clock If SKL = 0, SK = 0
1	0	Shift Register	Input to Shift Register	Serial Out	If SKL = 1, SK = Clock If SKL = 0, SK = 0
0	1	Not Used	Not Used	0	If SKL = 1, SK = 1 If SKL = 0, SK = 0
1	1	Not Used	Not Used	1	If SKL = 1, SK = 1 If SKL = 0, SK = 0

### COP420C/421C and COP430C/321C Instruction Set

Table 1 is a symbol table providing internal architecture, instruction operand, and operational symbols used in the instruction set table.

Table 2 provides the mnemonic, operand, machine code, data flow, skip conditions, and description associated with each instruction in the COP420C/421C/320C/321C instruction set.

**Table 1. COP420C/421C and COP320C/321C Instruction Set Table Symbols**

Symbol	Definition	Symbol	Definition
<b>INTERNAL ARCHITECTURE SYMBOLS</b>		<b>INSTRUCTION OPERAND SYMBOLS</b>	
A	4-bit Accumulator	d	4-bit Operand Field, 0-15 binary (RAM Digit Select)
B	6-bit RAM Address Register	r	2-bit Operand Field, 0-3 binary (RAM Register Select)
Br	Upper 2 bits of B (register address)	a	9-bit Operand Field, 0-511 binary (ROM Address)
Bd	Lower 4 bits of B (digit address)	y	4-bit Operand Field, 0-15 binary (Immediate Data)
C	1-bit Carry Register	RAM(s)	Contents of RAM location addressed by s
D	4-bit Data Output Port	ROM(t)	Contents of ROM location addressed by t
EN	4-bit Enable Register	<b>OPERATIONAL SYMBOLS</b>	
G	4-bit Register to latch data for G I/O Port	+	Plus
IL	Two 1-bit Latches Associated with the IN <sub>3</sub> or IN <sub>0</sub> inputs	-	Minus
IN	4-bit Input port	→	Replaces
L	8-bit TRI-STATE® I/O Port	↔	Is exchanged with
M	4-bit contents of RAM Memory pointed to by B Register	=	Is equal to
PC	10-bit ROM Address Register (program counter)	$\bar{A}$	The one's complement of A
Q	8-bit Register to latch data for L I/O Port	⊕	Exclusive-OR
SA	10-bit Subroutine Save Register A	:	Range of values
SB	10-bit Subroutine Save Register B		
SC	10-bit Subroutine Save Register C		
SIO	4-bit Shift Register and Counter		
SK	Logic-Controlled Clock Output		

**Table 2. COP420C/421C, COP320C/321C Instruction Set Table (Note 1)**

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>ARITHMETIC INSTRUCTIONS</b>						
ASC		30	0011 0000	A + C + RAM(B) → A Carry → C	Carry	Add with Carry, Skip on Carry
ADD		31	0011 0001	A + RAM(B) → A	None	Add RAM to A
ADT		4A	0100 1010	A + 10 <sub>10</sub> → A	None	Add Ten to A
AISC	y	5-	0101  y	A + y → A	Carry	Add Immediate, Skip on Carry (y ≠ 0)
CASC		10	0001 0000	$\bar{A}$ + RAM(B) + C → A Carry → C	Carry	Complement and Add with Carry, Skip on Carry
CLRA		00	0000 0000	0 → A	None	Clear A
COMP		40	0100 0000	$\bar{A}$ → A	None	One's complement of A to A
NOP		44	0100 0100	None	None	No Operation
RC		32	0011 0010	"0" → C	None	Reset C
SC		22	0010 0010	"1" → C	None	Set C
XOR		02	0000 0010	A ⊕ RAM(B) → A	None	Exclusive-OR RAM with A

Table 2. COP420C/421C, COP320C/321C Instruction Set Table (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
TRANSFER OF CONTROL INSTRUCTIONS						
JID		FF	$\boxed{1111 1111}$	ROM (PC <sub>9:8</sub> , A, M) → PC <sub>7:0</sub>	None	Jump Indirect (Note 3)
JMP	a	6-	$\boxed{0110 00 a_{9:8}}$ $\boxed{a_{7:0}}$	a → PC	None	Jump
JP	a	--	$\boxed{1 a_{6:0}}$ (pages 2,3 only) or $\boxed{11 a_{5:0}}$ (all other pages)	a → PC <sub>6:0</sub> a → PC <sub>5:0</sub>	None	Jump within Page (Note 4)
JSRP	a	--	$\boxed{10 a_{5:0}}$	PC + 1 → SA → SB → SC 0010 → PC <sub>9:6</sub> a → PC <sub>5:0</sub>	None	Jump to Subroutine Page (Note 5)
JSR	a	6-	$\boxed{0110 10 a_{9:8}}$ $\boxed{a_{7:0}}$	PC + 1 → SA → SB → SC a → PC	None	Jump to Subroutine
RET		48	$\boxed{0100 1000}$	SC → SB → SA → PC	None	Return from Subroutine
RETSK		49	$\boxed{0100 1001}$	SC → SB → SA → PC	Always Skip on Return	Return from Subroutine then Skip
IT		33 39	$\boxed{0011 0011}$ $\boxed{0011 1001}$	PC → PC		Idle till Timer overflows then continue
MEMORY REFERENCE INSTRUCTIONS						
CAMQ		33 3C	$\boxed{0011 0011}$ $\boxed{0011 1100}$	A → Q <sub>7:4</sub> RAM(B) → Q <sub>3:0</sub>	None	Copy A, RAM to Q
CQMA		33 3C	$\boxed{0011 0011}$ $\boxed{0011 1100}$	Q <sub>7:4</sub> → RAM(B) Q <sub>3:0</sub> → A	None	Copy Q to RAM, A
LD	r	-5	$\boxed{00 r 0101}$	RAM(B) → A Br ⊕ r → Br	None	Load RAM into A, Exclusive-OR Br with r
LDD	r,d	23 --	$\boxed{0010 0011}$ $\boxed{00 r d}$	RAM(r,d) → A	None	Load A with RAM pointed to directly by r,d
LQID		BF	$\boxed{1011 1111}$	ROM(PC <sub>9:8</sub> , A, M) → Q SB → SC	None	Load Q Indirect (Note 3)
RMB	0 1 2 3	4C 45 42 43	$\boxed{0100 1100}$ $\boxed{0100 0101}$ $\boxed{0100 0010}$ $\boxed{0100 0011}$	0 → RAM(B) <sub>0</sub> 0 → RAM(B) <sub>1</sub> 0 → RAM(B) <sub>2</sub> 0 → RAM(B) <sub>3</sub>	None	Reset RAM Bit
SMB	0 1 2 3	4D 47 46 4B	$\boxed{0100 1101}$ $\boxed{0100 1101}$ $\boxed{0100 0110}$ $\boxed{0100 1011}$	1 → RAM(B) <sub>0</sub> 1 → RAM(B) <sub>1</sub> 1 → RAM(B) <sub>2</sub> 1 → RAM(B) <sub>3</sub>	None	Set RAM Bit

Table 2. COP420C/421C, COP320C/321C Instruction Set Table (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description																
MEMORY REFERENCE INSTRUCTIONS (continued)																						
STII	y	7-	<table border="1"><tr><td>0</td><td>1</td><td>1</td><td>1</td></tr><tr><td>y</td><td></td><td></td><td></td></tr></table>	0	1	1	1	y				y → RAM(B) Bd + 1 → Bd	None	Store Memory Immediate and Increment Bd								
0	1	1	1																			
y																						
X	r	-6	<table border="1"><tr><td>0</td><td>0</td><td>r</td><td>0</td><td>1</td><td>1</td><td>0</td></tr></table>	0	0	r	0	1	1	0	RAM(B) ↔ A Br ⊕ r → Br	None	Exchange RAM with A, Exclusive-OR Br with r									
0	0	r	0	1	1	0																
XAD	r,d	23 --	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>r</td><td></td><td>d</td><td></td><td></td><td></td></tr></table>	0	0	1	0	0	0	1	1	1	0	r		d				RAM(r,d) ↔ A	None	Exchange A with RAM pointed to directly by r,d
0	0	1	0	0	0	1	1															
1	0	r		d																		
XDS	r	-7	<table border="1"><tr><td>0</td><td>0</td><td>r</td><td>0</td><td>1</td><td>1</td><td>1</td></tr></table>	0	0	r	0	1	1	1	RAM(B) ↔ A Bd - 1 → Bd Br ⊕ r → Br	Bd decrements past 0	Exchange RAM with A and Decrement Bd, Exclusive-OR Br with r									
0	0	r	0	1	1	1																
XIS	r	-4	<table border="1"><tr><td>0</td><td>0</td><td>r</td><td>0</td><td>1</td><td>0</td><td>0</td></tr></table>	0	0	r	0	1	0	0	RAM(B) ↔ A Bd + 1 → Bd Br ⊕ r → Br	Bd increments past 15	Exchange RAM with A and Increment Bd, Exclusive-OR Br with r									
0	0	r	0	1	0	0																
REGISTER REFERENCE INSTRUCTIONS																						
CAB		50	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	1	0	1	0	0	0	0	A → Bd	None	Copy A to Bd								
0	1	0	1	0	0	0	0															
CBA		4E	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr></table>	0	1	0	0	1	1	1	0	Bd → A	None	Copy Bd to A								
0	1	0	0	1	1	1	0															
LBI	r,d	--	<table border="1"><tr><td>0</td><td>0</td><td>r</td><td>(d-1)</td></tr></table> (d = 0, 9:15) or <table border="1"><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>r</td><td>d</td></tr></table> (any d)	0	0	r	(d-1)	0	0	1	0	0	1	1	1	0	r	d	r,d → B	Skip until not a LBI	Load B Immediate with r,d (Note 6)	
0	0	r	(d-1)																			
0	0	1	0	0	1	1																
1	0	r	d																			
LEI	y	33 6-	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>y</td><td></td><td></td><td></td></tr></table>	0	0	1	1	0	0	1	1	0	1	1	0	y				y → EN	None	Load EN Immediate (Note 7)
0	0	1	1	0	0	1	1															
0	1	1	0	y																		
XABR		12	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table>	0	0	0	1	0	0	1	0	A ↔ Br (0,0 → A <sub>3</sub> ,A <sub>2</sub> )	None	Exchange A with Br								
0	0	0	1	0	0	1	0															
TEST INSTRUCTIONS																						
SKC		20	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	1	0	0	0	0	0		C = "1"	Skip if C is True								
0	0	1	0	0	0	0	0															
SKE		21	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	1	0	0	0	0	1		A = RAM(B)	Skip if A Equals RAM								
0	0	1	0	0	0	0	1															
SKGZ		33 21	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	1	1	0	0	1	1	0	0	1	0	0	0	0	1		G <sub>3:0</sub> = 0	Skip if G is Zero (all 4 bits)
0	0	1	1	0	0	1	1															
0	0	1	0	0	0	0	1															
SKGBZ		33	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	1	1	0	0	1	1	1st byte		Skip if G Bit is Zero								
0	0	1	1	0	0	1	1															
	0	01	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	0	0	0	0	0	1	} 2nd byte	G <sub>0</sub> = 0									
0	0	0	0	0	0	0	1															
	1	11	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	0	1	0	0	0	1		G <sub>1</sub> = 0									
0	0	0	1	0	0	0	1															
	2	03	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	0	0	0	0	1	1	G <sub>2</sub> = 0										
0	0	0	0	0	0	1	1															
	3	13	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	0	1	0	0	1	1	G <sub>3</sub> = 0										
0	0	0	1	0	0	1	1															
SKMBZ	0	01	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	0	0	0	0	0	1		RAM(B) <sub>0</sub> = 0	Skip if RAM Bit is Zero								
0	0	0	0	0	0	0	1															
	1	11	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	0	1	0	0	0	1		RAM(B) <sub>1</sub> = 0									
0	0	0	1	0	0	0	1															
	2	03	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	0	0	0	0	1	1		RAM(B) <sub>2</sub> = 0									
0	0	0	0	0	0	1	1															
	3	13	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	0	1	0	0	1	1		RAM(B) <sub>3</sub> = 0									
0	0	0	1	0	0	1	1															
SKT		41	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	1	0	0	0	0	0	1		A time-base counter overflow has occurred since last test	Skip on Timer (Note 3)								
0	1	0	0	0	0	0	1															

Table 2. COP420C/421C, COP320C/321C Instruction Set Table (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description		
INPUT/OUTPUT INSTRUCTIONS								
ING		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	G → A	None	Input G Ports to A
	0011	0011						
	2A	<table border="1"><tr><td>0010</td><td>1010</td></tr></table>	0010	1010				
0010	1010							
ININ		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	IN → A	None	Input IN Inputs to A (Note 2)
	0011	0011						
	28	<table border="1"><tr><td>0010</td><td>1000</td></tr></table>	0010	1000				
0010	1000							
INIL		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	IL <sub>3</sub> , "0", IL <sub>0</sub> → A	None	Input IL Latches to A (Note 3)
	0011	0011						
	29	<table border="1"><tr><td>0010</td><td>1001</td></tr></table>	0010	1001				
0010	1001							
INL		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	L <sub>7:4</sub> → RAM(B) L <sub>3:0</sub> → A	None	Input L Ports to RAM,A
	0011	0011						
	2E	<table border="1"><tr><td>0010</td><td>1110</td></tr></table>	0010	1110				
0010	1110							
OBD		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	Bd → D	None	Output Bd to D Outputs
	0011	0011						
	3E	<table border="1"><tr><td>0011</td><td>1110</td></tr></table>	0011	1110				
0011	1110							
OGI	y	33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	y → G	None	Output to G Ports Immediate
		0011	0011					
5-	<table border="1"><tr><td>0101</td><td>y</td></tr></table>	0101	y					
0101	y							
OMG		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	RAM(B) → G	None	Output RAM to G Ports
	0011	0011						
	3A	<table border="1"><tr><td>0011</td><td>1010</td></tr></table>	0011	1010				
0011	1010							
XAS		4F	<table border="1"><tr><td>0100</td><td>1111</td></tr></table>	0100	1111	A ↔ SIO, C → SKL	None	Exchange A with SIO (Note 3)
0100	1111							

**Note 1:** All subscripts for alphabetical symbols indicate bit numbers unless explicitly defined (e.g., Br and Bd are explicitly defined). Bits are numbered 0 to N where 0 signifies the least significant bit (low-order, right-most bit). For example, A<sub>3</sub> indicates the most significant (left-most) bit of the 4-bit A register.

**Note 2:** The ININ instruction is not available on the 24-pin COP421C since this device does not contain the IN inputs.

**Note 3:** For additional information on the operation of the XAS, JID, LQID, INIL, and SKT instructions, see below.

**Note 4:** The JP instruction allows a jump, while in subroutine pages 2 or 3, to any ROM location within the two-page boundary of pages 2 or 3. The JP instruction, otherwise, permits a jump to a ROM location within the current 64-word page. JP may not jump to the last word of a page.

**Note 5:** A JSRP transfers program control to subroutine page 2 (0010 is loaded into the upper 4 bits of P). A JSRP may not be used when in pages 2 or 3. JSRP may not jump to the last word in page 2.

**Note 6:** LBI is a single-byte instruction if d = 0, 9, 10, 11, 12, 13, 14, or 15. The machine code for the lower 4 bits equals the binary value of the "d" data minus 1, e.g., to load the lower four bits of B (Bd) with the value 9 (1001<sub>2</sub>), the lower 4 bits of the LBI instruction equal 8 (1000<sub>2</sub>). To load 0, the lower 4 bits of the LBI instruction should equal 15 (1111<sub>2</sub>).

**Note 7:** Machine code for operand field y for LEI instruction should equal the binary value to be latched into EN, where a "1" or "0" in each bit of EN corresponds with the selection or deselection of a particular function associated with each bit. (See Functional Description, EN Register.)



## Interrupt

The following features are associated with the  $IN_1$  interrupt procedure and protocol and must be considered by the programmer when utilizing interrupts.

- The interrupt, once acknowledged as explained below, pushes the next sequential program counter address ( $PC + 1 \rightarrow SA \rightarrow SB \rightarrow SC$ ). Any previous contents of  $SC$  are lost. The program counter is set to hex address  $OFF$  (the last word of page 3) and  $EN_1$  is reset.
- An interrupt will be acknowledged only after the following conditions are met:
  - $EN_1$  has been set.
  - A low-going pulse ("1" to "0") of at least two instruction cycles wide occurs on the  $IN_1$  input.
  - A currently executing instruction has been completed.
  - All successive transfer of control instructions and successive LBIs have been completed (e.g., if the main program is executing a JP instruction which transfers program control to another JP instruction the interrupt will not be acknowledged until the second JP instruction has been executed).
- Upon acknowledgement of an interrupt, the skip logic status is saved and later restored upon the popping of the stack. For example, if an interrupt occurs during the execution of ASC (Add with Carry, Skip on Carry) instruction which results in carry, the skip logic status is saved and program control is transferred to the interrupt servicing routine at hex address  $OFF$ . At the *end* of the interrupt routine, a RET instruction is executed to "pop" the stack and return program control to the instruction following the original ASC. *At this time*, the skip logic is enabled and skips this instruction because of the previous ASC carry. Subroutines and the LQID instruction should not be nested within the interrupt servicing routine since their popping of the stack enables any previously saved main program skips, interfering with the orderly execution of the interrupt routine.
- The first instruction of the interrupt routine at hex address  $OFF$  must be a NOP.
- A LEI instruction can be put immediately before the RET to re-enable interrupts.

## MICROBUS™ Interface

The COP420C has an option which allows it to be used as a peripheral microprocessor device, inputting and outputting data from and to a host microprocessor ( $\mu P$ ).  $IN_1$ ,  $IN_2$ , and  $IN_3$  general purpose inputs become **MICROBUS™ compatible** read-strobe, chip-select, and write-strobe lines, respectively.  $IN_1$  becomes  $\overline{RD}$  — a logic "0" on this input will cause Q latch data to be enabled to the L ports for input to the  $\mu P$ .  $IN_2$  becomes  $\overline{CS}$  — a logic 0 selects the COP420C as a  $\mu P$  peripheral device and allows for the selection of one of several peripheral components.  $IN_3$  becomes  $\overline{WR}$  — a logic "0" on this line will write bus data from the L ports to the Q latches for input to the COP420C.  $G_0$  becomes  $INTR$  a "ready" output, reset by a write pulse from the  $\mu P$  on the  $\overline{WR}$  line, providing the "handshaking" capability necessary for asynchronous data transfer between the host CPU and the COP420C.

This option has been designed for compatibility with National's MICROBUS™ — a standard interconnect system for 8-bit parallel data transfer between MOS/LSI CPUs and interfacing devices. (See MICROBUS™, National Publication.) The functioning and timing relationships between the COP420C signal lines affected by this option are as specified for the MICROBUS™ interface, and are given in the AC electrical characteristics and shown in the timing diagrams (Figures 4 and 5). Connection of the COP420C to the MICROBUS™ is shown in Figure 6.

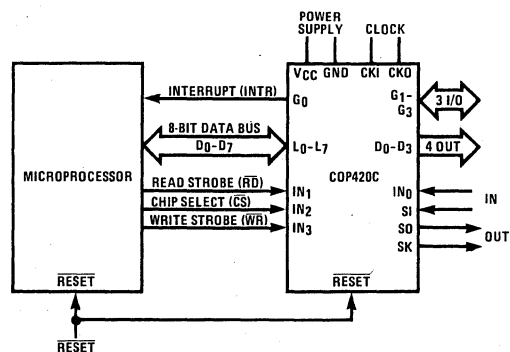


Figure 6. MICROBUS™ Option Interconnect

## Initialization

The Reset Logic, internal to the COP420C/421C, will initialize (clear) the device upon power-up if the power supply rise time is less than 1ms and greater than  $1\mu s$ . If the power supply rise time is greater than 1ms, the user must provide an external RC network and diode to the RESET pin as shown below. The RESET pin is configured as a Schmitt trigger input. If not used it should be connected to  $V_{CC}$ . Initialization will occur whenever a logic "0" is applied to the RESET input, provided it stays low for at least three instruction cycle times.

Upon initialization, the PC register is cleared to 0 (ROM address 0) and the A, B, C, D, EN, and G registers are cleared. The SK output is enabled as a SYNC clock, providing a pulse each instruction cycle time. *Data Memory (RAM) is not cleared upon initialization.* The first instruction at address 0 must be a CLRA.

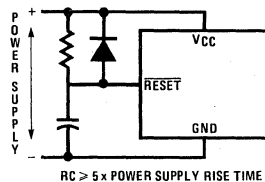


Figure 7. Power-Up Clear Circuit

The following information is provided to assist the user in understanding the operation of several unique instructions and to provide notes useful to programmers in writing COP420C/421C programs.

### XAS Instruction

XAS (Exchange A with SIO) exchanges the 4-bit contents of the accumulator with the 4-bit contents of the SIO register. The contents of SIO will contain serial-in/serial-out shift register data. An XAS instruction will also affect the SK output, providing a logic controlled clock. An XAS instruction must be performed once every 4 instruction cycles to effect a continuous data stream.

### JID Instruction

JID (Jump Indirect) is an indirect addressing instruction, transferring program control to a new ROM location pointed to indirectly by A and M. It loads the lower 8 bits of the ROM address register PC with the contents of ROM addressed by the 10-bit word, PC<sub>9,8</sub>, A, M. PC<sub>9</sub> and PC<sub>8</sub> are not affected by this instruction.

Note that JID requires 2 instruction cycles to execute.

### INIL Instruction

INIL (Input IL Latches to A) inputs 2 latches, IL<sub>3</sub> and IL<sub>0</sub> (see Figure 8) and CKO into A. The IL<sub>3</sub> and IL<sub>0</sub> latches are set if a low-going pulse ("1" to "0") has occurred on the IN<sub>3</sub> and IN<sub>0</sub> inputs since the last INIL instruction, provided the input pulse stays low for at least two instruction times. Execution of an INIL inputs IL<sub>3</sub> and IL<sub>0</sub> into A<sub>3</sub> and A<sub>0</sub> respectively, and resets these latches to allow them to respond to subsequent low-going pulses on the IN<sub>3</sub> and IN<sub>0</sub> lines. If CKO is mask programmed as a general purpose input, an INIL will input the state of CKO into A<sub>2</sub>. If CKO has not been so programmed, a "1" will be placed in A<sub>2</sub>. A "0" is always placed in A<sub>1</sub> upon the execution of an INIL. The general purpose inputs IN<sub>3</sub>, IN<sub>0</sub> are input to A upon execution of an ININ instruction. (See Table 2, ININ instruction.) INIL is useful in recognizing pulses of short duration or pulses which occur too often to be read conveniently by an ININ instruction. Note that IL latches are not cleared on reset. IL latches are not available on the COP421C.

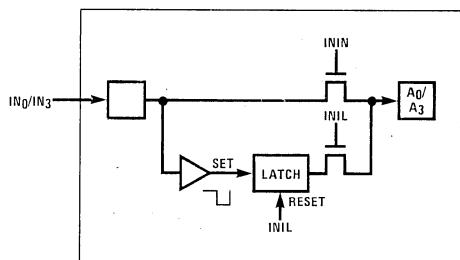


Figure 8. INIL Hardware Implementation

### LQID Instruction

LQID (Load Q Indirect) loads the 8-bit Q register with the contents of ROM pointed to by the 10-bit word PC<sub>9</sub>, PC<sub>8</sub>, A, M. LQID can be used for table lookup or code conversion such as BCD to seven-segment. The LQID instruction "pushes" the stack (PC + 1 → SA → SB → SC) and replaces the least significant 8 bits of PC as follows: A → PC<sub>7,4</sub>, RAM(B) → PC<sub>3,0</sub>, leaving PC<sub>9</sub> and PC<sub>8</sub> unchanged. The ROM data pointed to by the new address is fetched and loaded into the Q latches. Next, the stack is "popped" (SC → SB → SA → PC), restoring the saved value of PC to continue sequential program execution. Since LQID pushes SB → SC, the previous contents of SC are lost. Also, when LQID pops the stack, the previously pushed contents of SB are left in SC. The net result is that the contents of SB are placed in SC (SB → SC). Note that LQID takes two instruction cycle times to execute.

### SKT Instruction

The SKT (Skip On Timer) instruction tests the state of an internal 10-bit time base counter. This counter divides the instruction cycle clock frequency by 1024 and provides a latched indication of counter overflow. The SKT instruction tests this overflow latch, executing the next program instruction if the latch is not set. If the latch has been set since the previous test, the next program instruction is skipped and the latch is reset. The features associated with this instruction, therefore, allow the COP420C/421C to generate its own time base for real-time processing rather than relying on an external input signal.

For example, using a 32kHz watch crystal for the oscillator, the counter pulse frequency will be 4Hz. For time-of-day or similar real-time processing, the SKT instruction can call a routine which increments a "seconds" counter every 4 ticks.

### IT Instruction

The user may choose to use the IT function instead of the SKT function. The IT (Idle till Timer) instruction halts the processor and puts it in an idle state. This idle state reduces current drain since all logic (except the oscillator and time base counter) is stopped. The time base counter always divides CKI by 8192 regardless of the divide-by option selected (see Figures 10 and 11).

If in the divide-by-8 mode, the chip will come out of the idle state when the time base counter overflows. If in the divide-by-16 mode, the chip will come out of the idle state after the time base counter overflows TWICE. The IT instruction cannot be used in the divide-by-32 mode.

Therefore, the number of instruction cycles that the chip remains in the idle state is the SAME for both divide-by-8 and divide-by-16. For example, if CKI is 262kHz (divide-by-16) or is 131kHz (divide-by-8), the chip will come out of idle 16 times per second.

If using the dual clock feature, the user MUST switch the processor to the CKI oscillator (D0 = 0) before executing the IT instruction.

**Note:** If using the dual clock feature or the IT instruction, contact the factory for emulation assistance.

### Using Both SKT and IT Instructions

If specific guidelines are adhered to, the SKT instruction may be used when the IT instruction is enabled. (option 31 = 4 to 7).

1. Use divide by 8 CKI (option 3 = 1).
2. If using Dual clock, execute SKT only when operating from CKI clock.
3. After executing an SKT which gives a skip, execute another SKT instruction.

#### Sample Code:

```

SKT          ; test timer
JP          NO      ; no overflow
SKT          ; do another SKT
NOP         ; defeat skip
YES:        ; process timer overflow
:
:
:          (continue program)
:
NO:         ; no timer overflow, continue
    
```

Using this technique, a careful programmer can use both SKT and IT.

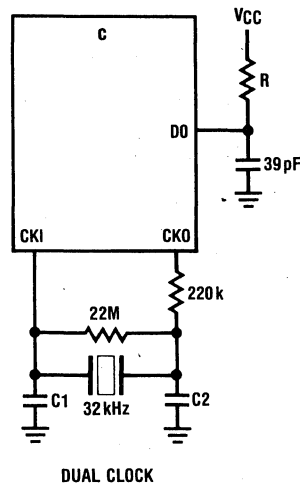
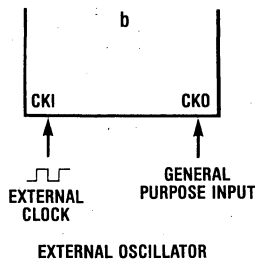
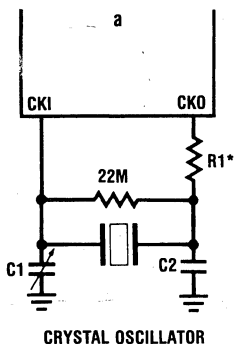
### Oscillator

There are three basic clock oscillator configurations available as shown by Figure 9.

- a. Crystal Controlled Oscillator.** CKI and CKO are connected to an external crystal (or resonator). The instruction cycle time equals the crystal frequency divided by 32, 16 or 8.
- b. External Oscillator.** CKI is connected to an external clock input signal. CKO is now available to be used as a general purpose input.
- c. Dual Oscillator.** By selecting the dual clock option, pin D0 is now a clock input. The user may connect a 32 kHz watch crystal to CKI and CKO and up to a 500 kHz clock to D0 (R/C or external). The user may then software select between the D0 oscillator for faster processing (D0 = 1) or the crystal for minimum current drain (D0 = 0). The time-base counter runs off the CKI oscillator even when the user selects D0 as the clock. Thus, a real time clock can be maintained by the IT instruction even when running off the RC oscillator. The SKT instruction is restricted when using the dual clock feature.

### CKO Pin Options

In a crystal controlled oscillator system, CKO is used as an output to the crystal network. As an option CKO can be a general purpose input, read into bit 2 of A (accumulator) upon execution of an INIL instruction.



Crystal Oscillator

Crystal Value	Component Values		
	R1*	C1	C2
2.097MHz	1k	5-36pF	30pF
32kHz	220k	5-36pF	30pF
500kHz	4.7k	47pF	82pF

\*Selected based on Crystal used.

R/C Oscillator

R (Ω)	V <sub>CC</sub>	Instruction Cycle Time
120k ± 10%	4.5 to 5.5V	15μs to 60μs
160k	3V	30μs to 120μs
180k	2.4V	50μs to 200μs

Figure 9.

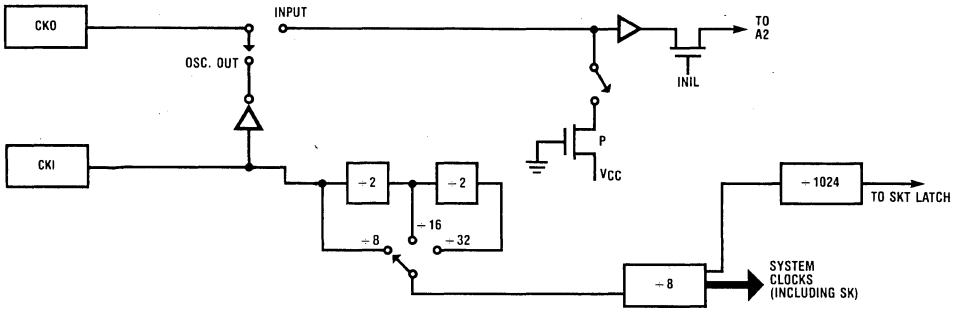


Figure 10a. Oscillator Options Block Diagram Using SKT Instruction

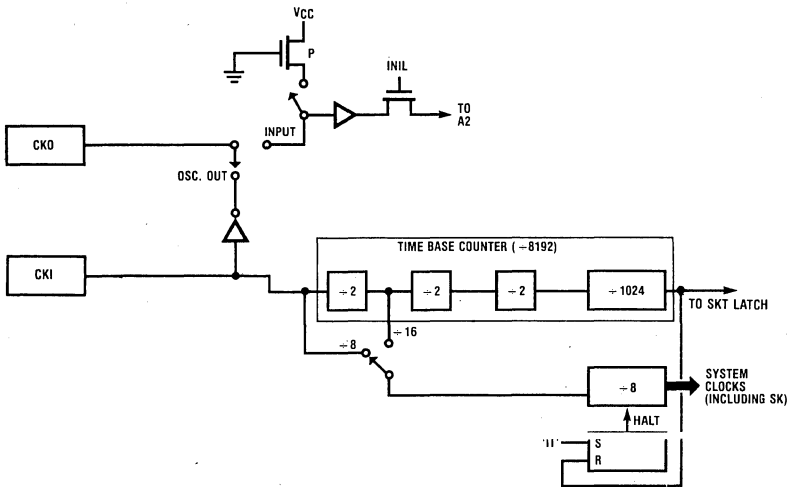


Figure 10b. Oscillator Options Block Diagram Using IT Instruction

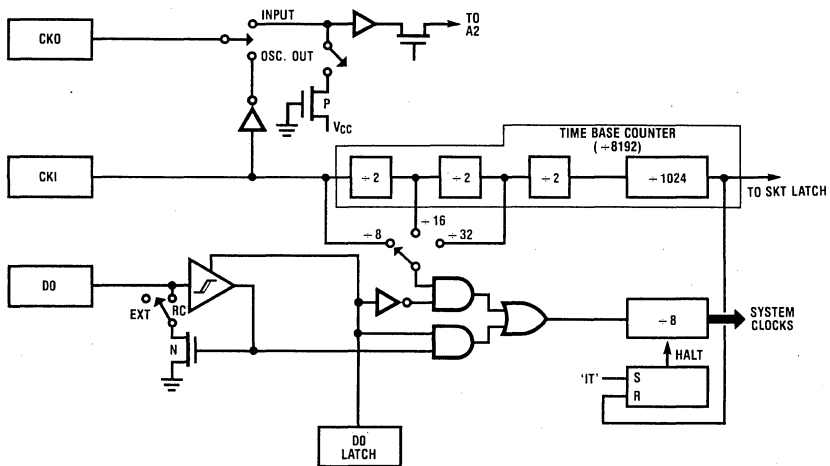


Figure 11. Dual Clock Option Block Diagram

## I/O Options

COP420C/421C outputs have the following optional configurations, illustrated in Figure 13:

- Standard** — An N-channel device to ground in conjunction with a P-channel device to  $V_{CC}$ , compatible with CMOS and LSTTL.
- Open-Drain** — An N-channel device to ground only, allowing external pull-up as required by the user's application.
- TRI-STATE<sup>®</sup> L Output** — A CMOS output buffer which may be disabled by program control. These outputs meet the requirements associated with the MICROBUS<sup>™</sup> option.
- Standard L Output** — This is the same configuration as c. above except that the sourcing current is standard.
- Open Drain L Output** — This has the N-channel device to ground only.

COP420C/421C inputs have the following options:

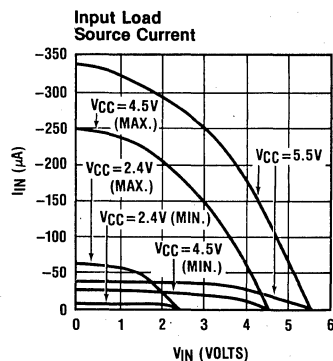
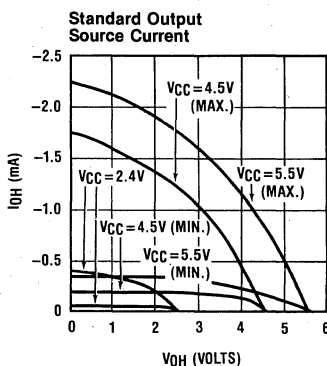
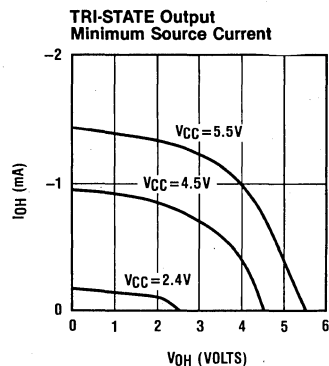
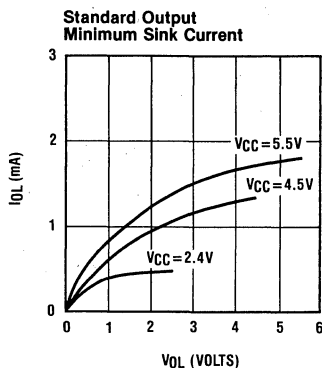
- An on-chip pullup load device to  $V_{CC}$ .
- A Hi-Z input which must be driven by user logic.

The above input and output configurations share common devices. Specifically, all configurations use one or more of four devices (numbered 1-4, respectively). Minimum and maximum current ( $I_{OUT}$  and  $V_{OUT}$ ) curves are given in Figure 12 for each of these devices to allow the designer to effectively use these I/O configurations.

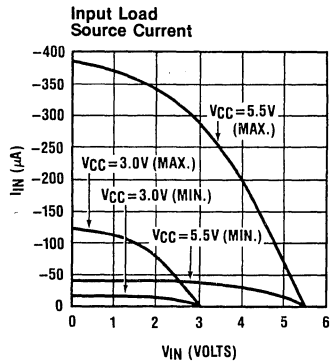
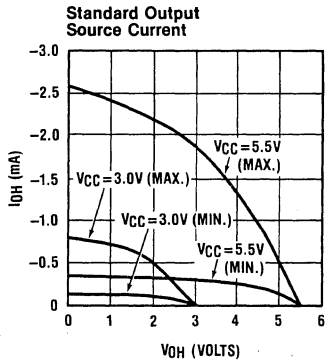
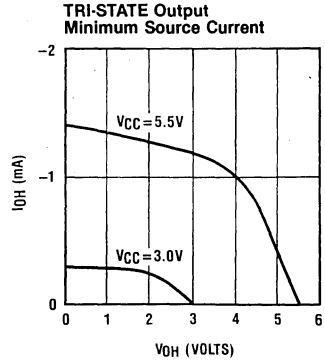
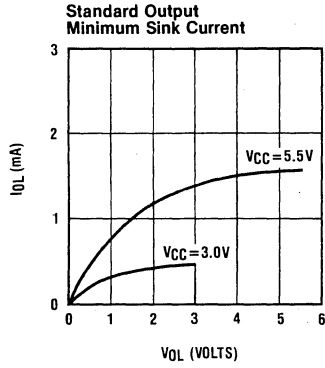
## COP421C

If the COP420C is bonded as a 24-pin device, it becomes the COP421C, illustrated in Figure 2, COP420C/421C Connection Diagrams. Note that the COP421C does not contain the four general purpose IN inputs ( $IN_3$ - $IN_0$ ). Use of this option precludes, of course, use of the IN options, interrupt feature, and the MICROBUS<sup>™</sup> option which uses  $IN_1$ - $IN_3$ . All other options are available for the COP421C.

## COP420C/421C I/O Characteristics



# COP320C/321C I/O Characteristics



**Instruction Set Notes**

- a. The first word of a program (ROM address 0) must be a CLRA (Clear A) instruction.
- b. Although skipped instructions are not executed, one instruction cycle time is devoted to skipping each byte of the skipped instruction. Thus all program paths except LQID or JID take the same number of cycle times whether instructions are skipped or executed. LQID and JID take two cycle times if executed and one if skipped.
- c. The ROM is organized into 16 pages of 64 words each. The Program Counter is an 10-bit binary counter, and will count through page boundaries. If a JP, JSRP, JID or LQID instruction is located in the last word of a page, the instruction operates as if it were in the next page. For example: a JP located in the last word of a page will jump to a location in the next page. Also, a LQID or JID located in the last word of page 3, 7, 11, or 15 will access data in the next group of 4 pages.

**COP420C Power Dissipation**

The lowest power configuration is at minimum voltage and lowest frequency. The user should take care that all inputs swing to full supply levels to insure that there are no DC current paths on inputs. An external square wave oscillator will use less current than a crystal or resonator since an input from a crystal is slow to transcend logic levels. For example: at 500 kHz, a crystal (or resonator)

will typically cause the 420C to draw 100µA more than with a square wave oscillator input. Power will increase with loading capacitance and frequency of the outputs.

The lowest possible current drain is when the processor is in the idle mode (see IT instruction).

Another method to reduce power is to use the dual clock option. The overall current drain will be an average of the low frequency current and the high frequency current, based on the amount of time spent at each frequency.

**COP420C TTL Interface**

The COP420C outputs can directly drive one standard LSTTL load. A pull-up device should be selected on inputs driven by TTL in order to bring the input signal up to the required logic "1" level.

**TEST MODE (Non-Standard Operation)**

The SO output has been configured to provide for standard test procedures for the custom-programmed COP420C. With SO forced to logic "1", two test modes are provided, depending upon the value of SI:

- a. RAM and Internal Logic Test Mode (SI = 1)
- b. ROM Test Mode (SI = 0)

These special test modes should not be employed by the user; they are intended for manufacturing test only.

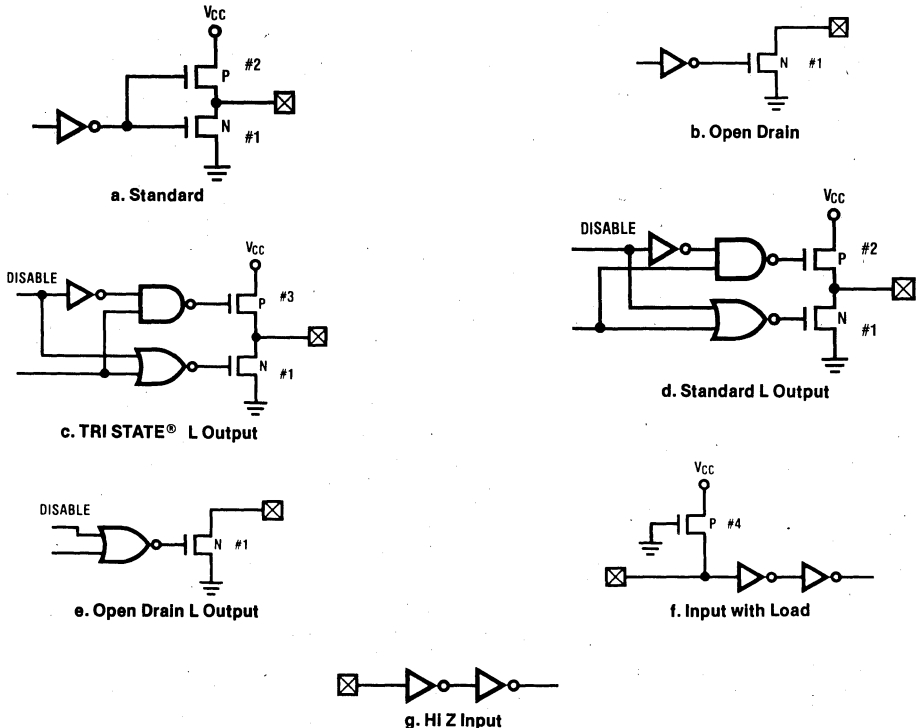


Figure 13. I/O Configurations

## Option List

The COP420C/320C mask-programmable options are assigned numbers which correspond with the COP420C pins.

The following is a list of COP420C options. When specifying a COP421C chip, Options 9, 10, 19, and 20 must all be set to zero. The options are programmed at the same time as the ROM pattern to provide the user with the hardware flexibility to interface to various I/O components using little or no external circuitry.

- Option 1 = 0: Ground Pin  
Not an option
- Option 2: CKO Output  
00 = Oscillator Output  
02 = General Input,  $V_{CC}$  Load  
04 = General Input, Hi-Z
- Option 3: CKI Input  
00 = Oscillator IN ( $\div 16$ )  
01 = Oscillator IN ( $\div 8$ )  
02 = Oscillator IN ( $\div 32$ )
- Option 4: RESET Input  
00 = Load  $V_{CC}$   
01 = Hi-Z
- Option 5: L<sub>7</sub> Driver  
00 = Standard Output  
01 = Open Drain  
02 = High Current TRI-STATE®
- Option 6: L<sub>6</sub> Driver  
same as Option 5
- Option 7: L<sub>5</sub> Driver  
same as Option 5
- Option 8: L<sub>4</sub> Driver  
same as Option 5
- Option 9: IN<sub>1</sub> Input  
00 = Load  $V_{CC}$   
01 = Hi-Z
- Option 10: IN<sub>2</sub> Input  
same as Option 9
- Option 11:  $V_{CC}$  pin  
not an option
- Option 12: L<sub>3</sub> Driver  
same as Option 5
- Option 13: L<sub>2</sub> Driver  
same as Option 5
- Option 14: L<sub>1</sub> Driver  
same as Option 5
- Option 15: L<sub>0</sub> Driver  
same as Option 5
- Option 16: SI Input  
same as Option 9
- Option 17: SO Driver  
00 = Standard Output  
01 = Open Drain
- Option 18: SK Driver  
same as Option 17
- Option 19: IN<sub>0</sub> Input  
same as Option 9
- Option 20: IN<sub>3</sub> Input  
same as Option 9
- Option 21: G<sub>0</sub> I/O Port  
same as Option 17
- Option 22: G<sub>1</sub> I/O Port  
same as Option 17
- Option 23: G<sub>2</sub> I/O Port  
same as Option 17
- Option 24: G<sub>3</sub> I/O Port  
same as Option 17
- Option 25: D<sub>3</sub> Output  
same as Option 17
- Option 26: D<sub>2</sub> Output  
same as Option 17
- Option 27: D<sub>1</sub> Output  
same as Option 17
- Option 28: D<sub>0</sub> Output  
00 = Standard Output  
01 = Open Drain (or Dual Clock)
- Option 29: COP Function  
00 = Normal  
01 = MICROBUS™
- Option 30: COP Bonding  
00 = COP420C (28-pin package)  
01 = COP421C (24-pin package)  
02 = COP420C and COP421C, same ROM (same die purchased in both 24- and 28-pin versions)
- Option 31: Clock/Timer Mode  
02 = Xtal/Ext. Osc. in; SKT instruction enabled; no IT  
04\* = Xtal/Ext. Osc. in; IT instruction enabled; SKT restricted  
06\* = Xtal/Ext. Osc. in; Dual Clock (RC); IT enabled  
SKT restricted  
07\* = Xtal/Ext. Osc. in; Dual Clock (Ext.) IT enabled;  
SKT restricted

\*Contact factory for emulation assistance. Also note KIL maximum frequency specifications (page 3).





# COP420L/COP421L/COP422L and COP320L/COP321L/COP322L Single-Chip N-Channel Microcontrollers

## General Description

The COP420L, COP421L, COP422L, COP320L, COP321L, and COP322L Single-Chip N-Channel Microcontrollers are members of the COPS™ family, fabricated using N-channel, silicon gate MOS technology. These controller oriented processors are complete microcomputers containing all system timing, internal logic, ROM, RAM, and I/O necessary to implement dedicated control functions in a variety of applications. Features include single supply operation, a variety of output configuration options, with an instruction set, internal architecture, and I/O scheme designed to facilitate keyboard input, display output, and BCD data manipulation. The COP421L and COP422L are identical to the COP420L, but with 19 and 15 I/O lines, respectively, instead of 23. They are an appropriate choice for use in numerous human interface control environments. Standard test procedures and reliable high-density fabrication techniques provide the medium to large volume customers with a customized controller oriented processor at a low end-product cost.

The COP320L, COP321L, and COP322L are exact functional equivalents, but extended temperature range versions, of the COP420L, COP421L, and COP422L respectively.

COPS and MICROWIRE are trademarks of National Semiconductor Corp. TRI-STATE is a registered trademark of National Semiconductor Corp.

## Features

- Low cost
- Powerful instruction set
- 1k × 8 ROM, 64 × 4 RAM
- 23 I/O lines (COP420L)
- True vectored interrupt, plus restart
- Three-level subroutine stack
- 16μs instruction time
- Single supply operation (4.5–6.3V)
- Low current drain (8mA max.)
- Internal time-base counter for real-time processing
- Internal binary counter register with MICROWIRE™ compatible serial I/O
- General purpose and TRI-STATE® outputs
- LSTTL/CMOS compatible in and out
- Direct drive of LED digit and segment lines
- Software/hardware compatible with other members of COP400 family
- Extended temperature range device COP320L/COP321L/COP322L (–40°C to +85°C)
- Wider supply range (4.5V – 9.5V) optionally available

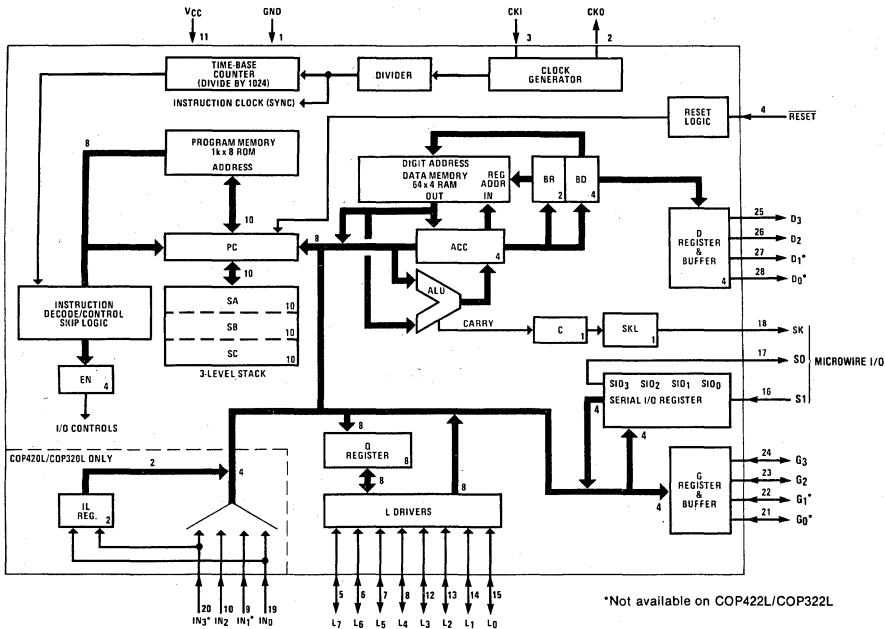


Figure 1. COP420L/COP421L/COP422L, COP320L/COP321L/COP322L Block Diagram

**COP420L/COP421L/COP422L****Absolute Maximum Ratings**

Voltage at Any Pin Relative to GND	-0.5V to +10V
Ambient Operating Temperature	0°C to +70°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C
Power Dissipation	
COP420L/COP421L	0.75 Watt at 25°C 0.4 Watt at 70°C
COP422L	0.65 Watt at 25°C 0.3 Watt at 70°C
Total Source Current	120 mA
Total Sink Current	120 mA

*Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

**DC Electrical Characteristics**  $0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{\text{CC}} \leq 9.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Standard Operating Voltage ( $V_{\text{CC}}$ )	Note 1	4.5	6.3	V
Optional Operating Voltage ( $V_{\text{CC}}$ )		4.5	9.5	V
Power Supply Ripple	peak to peak		0.5	V
Operating Supply Current	all inputs and outputs open		9	mA
Input Voltage Levels				
CKI Input Levels				
Crystal Input ( $\pm 32, \pm 16, \pm 8$ )				
Logic High ( $V_{\text{IH}}$ )		2.0		V
Logic Low ( $V_{\text{IL}}$ )		-0.3	0.4	V
Schmitt Trigger Input ( $\div 4$ )				
Logic High ( $V_{\text{IH}}$ )		$0.7V_{\text{CC}}$		V
Logic Low ( $V_{\text{IL}}$ )		-0.3	0.6	V
RESET Input Levels	Schmitt Trigger Input			
Logic High		$0.7V_{\text{CC}}$		V
Logic Low		-0.3	0.6	V
SO Input Level (Test mode)		2.0	2.5	V
All Other Inputs				
Logic High	$V_{\text{CC}} = \text{Max.}$	3.0		V
Logic High	with TTL trip level options	2.0		V
Logic Low	selected, $V_{\text{CC}} = 5\text{V} \pm 5\%$	-0.3	0.8	V
Logic High	with high trip level options	3.6		V
Logic Low	selected	-0.3	1.2	V
Input Capacitance			7	pF
Hi-Z Input Leakage		-1	+1	$\mu\text{A}$
Output Voltage Levels				
LSTTL Operation	$V_{\text{CC}} = 5\text{V} \pm 5\%$			
Logic High ( $V_{\text{OH}}$ )	$I_{\text{OH}} = -25\mu\text{A}$	2.7		V
Logic Low ( $V_{\text{OL}}$ )	$I_{\text{OL}} = 0.36\text{mA}$		0.4	V
CMOS Operation				
Logic High	$I_{\text{OH}} = -10\mu\text{A}$	$V_{\text{CC}} - 1$		V
Logic Low	$I_{\text{OL}} = +10\mu\text{A}$		0.2	V

**Note 1:**  $V_{\text{CC}}$  voltage change must be less than 0.5V in a 1ms period to maintain proper operation.

## COP420L/COP421L/COP422L

DC Electrical Characteristics (continued)  $0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{\text{CC}} \leq 9.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Output Current Levels				
Output Sink Current				
SO and SK Outputs ( $I_{\text{OL}}$ )	$V_{\text{CC}} = 9.5\text{V}$ , $V_{\text{OL}} = 0.4\text{V}$	1.8		mA
	$V_{\text{CC}} = 6.3\text{V}$ , $V_{\text{OL}} = 0.4\text{V}$	1.2		mA
	$V_{\text{CC}} = 4.5\text{V}$ , $V_{\text{OL}} = 0.4\text{V}$	0.9		mA
$L_0$ - $L_7$ Outputs and Standard $G_0$ - $G_3$ , $D_0$ - $D_3$ Outputs ( $I_{\text{OL}}$ )	$V_{\text{CC}} = 9.5\text{V}$ , $V_{\text{OL}} = 0.4\text{V}$	0.8		mA
	$V_{\text{CC}} = 6.3\text{V}$ , $V_{\text{OL}} = 0.4\text{V}$	0.5		mA
	$V_{\text{CC}} = 4.5\text{V}$ , $V_{\text{OL}} = 0.4\text{V}$	0.4		mA
$G_0$ - $G_3$ and $D_0$ - $D_3$ Outputs with High Current Options ( $I_{\text{OL}}$ )	$V_{\text{CC}} = 9.5\text{V}$ , $V_{\text{OL}} = 1.0\text{V}$	15		mA
	$V_{\text{CC}} = 6.3\text{V}$ , $V_{\text{OL}} = 1.0\text{V}$	11		mA
	$V_{\text{CC}} = 4.5\text{V}$ , $V_{\text{OL}} = 1.0\text{V}$	7.5		mA
$G_0$ - $G_3$ and $D_0$ - $D_3$ Outputs with Very High Current Options ( $I_{\text{OL}}$ )	$V_{\text{CC}} = 9.5\text{V}$ , $V_{\text{OL}} = 1.0\text{V}$	30		mA
	$V_{\text{CC}} = 6.3\text{V}$ , $V_{\text{OL}} = 1.0\text{V}$	22		mA
	$V_{\text{CC}} = 4.5\text{V}$ , $V_{\text{OL}} = 1.0\text{V}$	15		mA
CKI (Single-pin RC oscillator)	$V_{\text{CC}} = 4.5\text{V}$ , $V_{\text{IH}} = 3.5\text{V}$	2		mA
CKO	$V_{\text{CC}} = 4.5\text{V}$ , $V_{\text{OL}} = 0.4\text{V}$	0.2		mA
Output Source Current				
Standard Configuration, All Outputs ( $I_{\text{OH}}$ )				
	$V_{\text{CC}} = 9.5\text{V}$ , $V_{\text{OH}} = 2.0\text{V}$	-140	-800	$\mu\text{A}$
	$V_{\text{CC}} = 6.3\text{V}$ , $V_{\text{OH}} = 2.0\text{V}$	-75	-480	$\mu\text{A}$
	$V_{\text{CC}} = 4.5\text{V}$ , $V_{\text{OH}} = 2.0\text{V}$	-30	-250	$\mu\text{A}$
Push-Pull Configuration SO and SK Outputs ( $I_{\text{OH}}$ )				
	$V_{\text{CC}} = 9.5\text{V}$ , $V_{\text{OH}} = 4.75\text{V}$	-1.4		mA
	$V_{\text{CC}} = 6.3\text{V}$ , $V_{\text{OH}} = 2.4\text{V}$	-1.4		mA
	$V_{\text{CC}} = 4.5\text{V}$ , $V_{\text{OH}} = 1.0\text{V}$	-1.2		mA
LED Configuration, $L_0$ - $L_7$ Outputs, Low Current Driver Option ( $I_{\text{OH}}$ )				
	$V_{\text{CC}} = 9.5\text{V}$ , $V_{\text{OH}} = 2.0\text{V}$	-1.5	-18	mA
	$V_{\text{CC}} = 6.0\text{V}$ , $V_{\text{OH}} = 2.0\text{V}$	-1.5	-13	mA
LED Configuration, $L_0$ - $L_7$ Outputs, High Current Driver Option ( $I_{\text{OH}}$ )				
	$V_{\text{CC}} = 9.5\text{V}$ , $V_{\text{OH}} = 2.0\text{V}$	-3.0	-35	mA
	$V_{\text{CC}} = 6.0\text{V}$ , $V_{\text{OH}} = 2.0\text{V}$	-3.0	-25	mA
TRI-STATE <sup>®</sup> Configuration, $L_0$ - $L_7$ Outputs, Low Current Driver Option ( $I_{\text{OH}}$ )				
	$V_{\text{CC}} = 9.5\text{V}$ , $V_{\text{OH}} = 5.5\text{V}$	-0.75		mA
	$V_{\text{CC}} = 6.3\text{V}$ , $V_{\text{OH}} = 3.2\text{V}$	-0.8		mA
	$V_{\text{CC}} = 4.5\text{V}$ , $V_{\text{OH}} = 1.5\text{V}$	-0.9		mA
TRI-STATE <sup>®</sup> Configuration, $L_0$ - $L_7$ Outputs, High Current Driver Option ( $I_{\text{OH}}$ )				
	$V_{\text{CC}} = 9.5\text{V}$ , $V_{\text{OH}} = 5.5\text{V}$	-1.5		mA
	$V_{\text{CC}} = 6.3\text{V}$ , $V_{\text{OH}} = 3.2\text{V}$	-1.6		mA
	$V_{\text{CC}} = 4.5\text{V}$ , $V_{\text{OH}} = 1.5\text{V}$	-1.8		mA
Input Load Source Current	$V_{\text{CC}} = 5.0\text{V}$ , $V_{\text{IL}} = 0\text{V}$	-10	-140	$\mu\text{A}$
CKO Output				
RAM Power Supply Option Power Requirement				
	$V_R = 3.3\text{V}$		3.0	mA
TRI-STATE <sup>®</sup> Output Leakage Current		-2.5	+2.5	$\mu\text{A}$
Total Sink Current Allowed				
All Outputs Combined				
D, G Ports			120	mA
$L_7$ - $L_4$			120	mA
$L_3$ - $L_0$			4	mA
All Other Pins			4	mA
Total Source Current Allowed				
All I/O Combined				
$L_7$ - $L_4$			1.5	mA
$L_3$ - $L_0$			120	mA
Each L Pin			60	mA
All Other Pins			30	mA
			1.5	mA

**COP320L/COP321L/COP322L****Absolute Maximum Ratings**

Voltage at Any Pin Relative to GND	-0.5V to +10V
Ambient Operating Temperature	-40°C to +85°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C
Power Dissipation	
COP320L/COP321L	0.75 Watt at 25°C 0.4 Watt at 85°C
COP322L	0.65 Watt at 25°C 0.20 Watt at 85°C
Total Source Current	120 mA
Total Sink Current	120 mA

*Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

**DC Electrical Characteristics**  $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{\text{CC}} \leq 7.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Standard Operating Voltage ( $V_{\text{CC}}$ )	Note 1	4.5	5.5	V
Optional Operating Voltage ( $V_{\text{CC}}$ )		4.5	7.5	V
Power Supply Ripple	peak to peak		0.5	V
Operating Supply Current	all inputs and outputs open		11	mA
Input Voltage Levels				
CKI Input Levels				
Crystal Input				
Logic High ( $V_{\text{IH}}$ )		2.2		V
Logic Low ( $V_{\text{IL}}$ )		-0.3	0.3	V
Schmitt Trigger Input				
Logic High ( $V_{\text{IH}}$ )		$0.7V_{\text{CC}}$		V
Logic Low ( $V_{\text{IL}}$ )		-0.3	0.4	V
RESET Input Levels	Schmitt Trigger Input			
Logic High		$0.7V_{\text{CC}}$		V
Logic Low		-0.3	0.4	V
SO Input Level (Test mode)		2.2	2.5	V
All Other Inputs				
Logic High	$V_{\text{CC}} = \text{Max.}$	3.0		V
Logic High	with TTL trip level options	2.2		V
Logic Low	selected, $V_{\text{CC}} = 5\text{V} \pm 5\%$	-0.3	0.6	V
Logic High	with high trip level options	3.6		V
Logic Low	selected	-0.3	1.2	V
Input Capacitance			7	pF
Hi-Z Input Leakage		-2	+2	$\mu\text{A}$
Output Voltage Levels				
LSTTL Operation	$V_{\text{CC}} = 5\text{V} \pm 5\%$			
Logic High ( $V_{\text{OH}}$ )	$I_{\text{OH}} = -20\mu\text{A}$	2.7		V
Logic Low ( $V_{\text{OL}}$ )	$I_{\text{OL}} = 0.36\text{mA}$		0.4	V
CMOS Operation				
Logic High	$I_{\text{OH}} = -10\mu\text{A}$	$V_{\text{CC}} - 1$		V
Logic Low	$I_{\text{OL}} = +10\mu\text{A}$		0.2	V

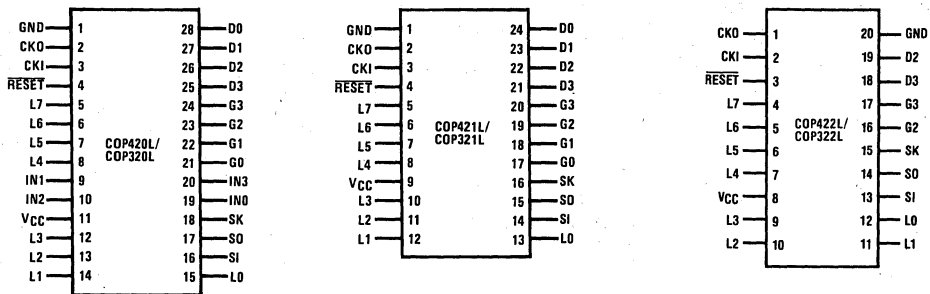
**Note 1:**  $V_{\text{CC}}$  voltage change must be less than 0.5V in a 1ms period to maintain proper operation.

**COP320L/COP321L/COP322L****DC Electrical Characteristics** (continued)  $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{\text{CC}} \leq 7.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
<b>Output Current Levels</b>				
<b>Output Sink Current</b>				
SO and SK Outputs ( $I_{\text{OL}}$ )	$V_{\text{CC}} = 7.5\text{V}$ , $V_{\text{OL}} = 0.4\text{V}$	1.4		mA
	$V_{\text{CC}} = 5.5\text{V}$ , $V_{\text{OL}} = 0.4\text{V}$	1.0		mA
	$V_{\text{CC}} = 4.5\text{V}$ , $V_{\text{OL}} = 0.4\text{V}$	0.8		mA
$L_0$ - $L_7$ Outputs and Standard	$V_{\text{CC}} = 7.5\text{V}$ , $V_{\text{OL}} = 0.4\text{V}$	0.6		mA
$G_0$ - $G_3$ , $D_0$ - $D_3$ Outputs ( $I_{\text{OL}}$ )	$V_{\text{CC}} = 5.5\text{V}$ , $V_{\text{OL}} = 0.4\text{V}$	0.5		mA
	$V_{\text{CC}} = 4.5\text{V}$ , $V_{\text{OL}} = 0.4\text{V}$	0.4		mA
$G_0$ - $G_3$ and $D_0$ - $D_3$ Outputs with High Current Options ( $I_{\text{OL}}$ )	$V_{\text{CC}} = 7.5\text{V}$ , $V_{\text{OL}} = 1.0\text{V}$	12		mA
	$V_{\text{CC}} = 5.5\text{V}$ , $V_{\text{OL}} = 1.0\text{V}$	9		mA
	$V_{\text{CC}} = 4.5\text{V}$ , $V_{\text{OL}} = 1.0\text{V}$	7		mA
$G_0$ - $G_3$ and $D_0$ - $D_3$ Outputs with Very High Current Options ( $I_{\text{OL}}$ )	$V_{\text{CC}} = 7.5\text{V}$ , $V_{\text{OL}} = 1.0\text{V}$	24		mA
	$V_{\text{CC}} = 5.5\text{V}$ , $V_{\text{OL}} = 1.0\text{V}$	18		mA
	$V_{\text{CC}} = 4.5\text{V}$ , $V_{\text{OL}} = 1.0\text{V}$	14		mA
CKI (Single-pin RC oscillator)	$V_{\text{CC}} = 4.5\text{V}$ , $V_{\text{IH}} = 3.5\text{V}$	2		mA
CKO	$V_{\text{CC}} = 4.5\text{V}$ , $V_{\text{OL}} = 0.4\text{V}$	0.2		mA
<b>Output Source Current</b>				
Standard Configuration, All Outputs ( $I_{\text{OH}}$ )	$V_{\text{CC}} = 7.5\text{V}$ , $V_{\text{OH}} = 2.0\text{V}$	-100	-900	$\mu\text{A}$
	$V_{\text{CC}} = 5.5\text{V}$ , $V_{\text{OH}} = 2.0\text{V}$	-55	-600	$\mu\text{A}$
	$V_{\text{CC}} = 4.5\text{V}$ , $V_{\text{OH}} = 2.0\text{V}$	-28	-350	$\mu\text{A}$
Push-Pull Configuration SO and SK Outputs ( $I_{\text{OH}}$ )	$V_{\text{CC}} = 7.5\text{V}$ , $V_{\text{OH}} = 3.75\text{V}$	-0.85		mA
	$V_{\text{CC}} = 5.5\text{V}$ , $V_{\text{OH}} = 2.0\text{V}$	-1.1		mA
	$V_{\text{CC}} = 4.5\text{V}$ , $V_{\text{OH}} = 1.0\text{V}$	-1.2		mA
LED Configuration, $L_0$ - $L_7$ Outputs, Low Current Driver Option ( $I_{\text{OH}}$ )	$V_{\text{CC}} = 7.5\text{V}$ , $V_{\text{OH}} = 2.0\text{V}$	-1.4	-27	mA
	$V_{\text{CC}} = 6.0\text{V}$ , $V_{\text{OH}} = 2.0\text{V}$	-1.4	-17	mA
	$V_{\text{CC}} = 5.5\text{V}$ , $V_{\text{OH}} = 2.0\text{V}$	-0.7	-15	mA
LED Configuration, $L_0$ - $L_7$ Outputs, High Current Driver Option ( $I_{\text{OH}}$ )	$V_{\text{CC}} = 7.5\text{V}$ , $V_{\text{OH}} = 2.0\text{V}$	-2.7	-54	mA
	$V_{\text{CC}} = 6.0\text{V}$ , $V_{\text{OH}} = 2.0\text{V}$	-2.7	-34	mA
	$V_{\text{CC}} = 5.5\text{V}$ , $V_{\text{OH}} = 2.0\text{V}$	-1.4	-30	mA
TRI-STATE <sup>®</sup> Configuration, $L_0$ - $L_7$ Outputs, Low Current Driver Option ( $I_{\text{OH}}$ )	$V_{\text{CC}} = 7.5\text{V}$ , $V_{\text{OH}} = 4.0\text{V}$	-0.7		mA
	$V_{\text{CC}} = 5.5\text{V}$ , $V_{\text{OH}} = 2.7\text{V}$	-0.6		mA
	$V_{\text{CC}} = 4.5\text{V}$ , $V_{\text{OH}} = 1.5\text{V}$	-0.9		mA
TRI-STATE <sup>®</sup> Configuration, $L_0$ - $L_7$ Outputs, High Current Driver Option ( $I_{\text{OH}}$ )	$V_{\text{CC}} = 7.5\text{V}$ , $V_{\text{OH}} = 4.0\text{V}$	-1.4		mA
	$V_{\text{CC}} = 5.5\text{V}$ , $V_{\text{OH}} = 2.7\text{V}$	-1.2		mA
	$V_{\text{CC}} = 4.5\text{V}$ , $V_{\text{OH}} = 1.5\text{V}$	-1.8		mA
Input Load Source Current	$V_{\text{CC}} = 5.0\text{V}$ , $V_{\text{IL}} = 0\text{V}$	-10	-200	$\mu\text{A}$
CKO Output				
RAM Power Supply Option Power Requirement	$V_R = 3.3\text{V}$		4.0	mA
TRI-STATE <sup>®</sup> Output Leakage Current		-5	+5	$\mu\text{A}$
<b>Total Sink Current Allowed</b>				
All Outputs Combined			120	mA
D, G Ports			120	mA
$L_7$ - $L_4$			4	mA
$L_3$ - $L_0$			4	mA
All Other Pins			1.5	mA
<b>Total Source Current Allowed</b>				
All I/O Combined			120	mA
$L_7$ - $L_4$			60	mA
$L_3$ - $L_0$			60	mA
Each L Pin			30	mA
All Other Pins			1.5	mA

**AC Electrical Characteristics**COP420L/COP421L/COP422L:  $0^{\circ}\text{C} \leq T_A \leq 70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 9.5\text{V}$  unless otherwise noted.COP320L/COP321L/COP322L:  $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 7.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Instruction Cycle Time — $t_C$		15	40	$\mu\text{s}$
CKI				
Input Frequency — $f_i$	$\div 32$ mode	0.8	2.1	MHz
	$\div 16$ mode	0.4	1.0	MHz
	$\div 8$ mode	0.2	0.5	MHz
	$\div 4$ mode	0.1	0.26	MHz
Duty Cycle		30	60	%
Rise Time	$f_i = 2\text{MHz}$		120	ns
Fall Time			80	ns
CKI Using RC ( $\div 4$ )	$R = 56\text{k}\Omega \pm 5\%$ $C = 100\text{pF} \pm 10\%$			
Instruction Cycle Time		15	28	$\mu\text{s}$
CKO as SYNC Input				
$t_{\text{SYNC}}$		400		ns
INPUTS:				
$\text{IN}_3 - \text{IN}_0, \text{G}_3 - \text{G}_0, \text{L}_7 - \text{L}_0$			8.0	$\mu\text{s}$
$t_{\text{SETUP}}$			1.3	$\mu\text{s}$
$t_{\text{HOLD}}$				
SI			2.0	$\mu\text{s}$
$t_{\text{SETUP}}$			1.0	$\mu\text{s}$
$t_{\text{HOLD}}$				
OUTPUT PROPAGATION DELAY	Test condition: $C_L = 50\text{pF}, R_L = 20\text{k}\Omega, V_{\text{OUT}} = 1.5\text{V}$			
SO, SK Outputs			4.0	$\mu\text{s}$
$t_{\text{pd1}}, t_{\text{pd0}}$				
All Other Outputs			5.6	$\mu\text{s}$
$t_{\text{pd1}}, t_{\text{pd0}}$				



Order Number COP420L/N, COP320L/N NS Package N28A      Order Number COP421L/N, COP321L/N NS Package N24A      Order Number COP422L/N, COP322L/N NS Package N20A

Figure 2. Connection Diagrams

Pin	Description	Pin	Description
L7-L <sub>0</sub>	8 bidirectional I/O ports with TRI-STATE®	SK	Logic-controlled clock (or general purpose output)
G <sub>3</sub> -G <sub>0</sub>	4 bidirectional I/O ports	CKI	System oscillator input
D <sub>3</sub> -D <sub>0</sub>	4 general purpose outputs	CKO	System oscillator output (or general purpose input, RAM power supply or SYNC input)
IN <sub>3</sub> -IN <sub>0</sub>	4 general purpose inputs (COP420L only)	RESET	System reset input
SI	Serial input (or counter input)	V <sub>CC</sub>	Power supply
SO	Serial output (or general purpose output)	GND	Ground

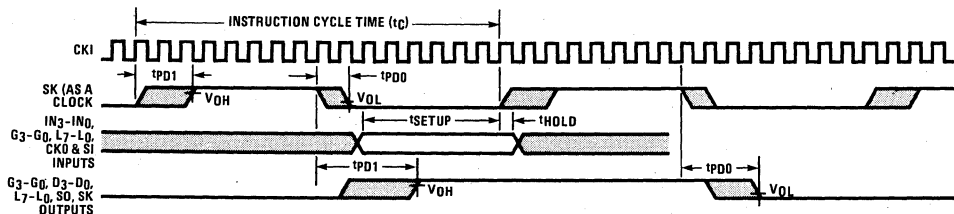


Figure 3. Input/Output Timing Diagrams (Crystal Divide-by-16 Mode)

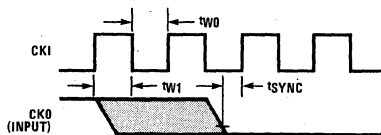


Figure 3a. Synchronization Timing

## Functional Description

For ease of reading this description, only COP420L and/or COP421L are referenced; however, all such references apply also to COP320L, COP321L, COP322L, or COP422L.

A block diagram of the COP420L is given in Figure 1. Data paths are illustrated in simplified form to depict how the various logic elements communicate with each other in implementing the instruction set of the device. Positive logic is used. When a bit is set, it is a logic "1" (greater than 2 volts). When a bit is reset, it is a logic "0" (less than 0.8 volts).

### Program Memory

Program Memory consists of a 1,024 byte ROM. As can be seen by an examination of the COP420L/421L instruction set, these words may be program instructions, program data or ROM addressing data. Because of the special characteristics associated with the JP, JSRP, JID and LQID instructions, ROM must often be thought of as being organized into 16 pages of 64 words each.

ROM addressing is accomplished by a 10-bit PC register. Its binary value selects one of the 1,024 8-bit words contained in ROM. A new address is loaded into the PC register during each instruction cycle. Unless the instruction is a transfer of control instruction, the PC register is loaded with the next sequential 10-bit binary count value. Three levels of subroutine nesting are implemented by the 10-bit subroutine save registers, SA, SB and SC, providing a last-in, first-out (LIFO) hardware subroutine stack.

ROM instruction words are fetched, decoded and executed by the Instruction Decode, Control and Skip Logic circuitry.

### Data Memory

Data memory consists of a 256-bit RAM, organized as 4 data registers or 16 4-bit digits. RAM addressing is implemented by a 6-bit B register whose upper 2 bits (Br) select 1 of 4 data registers and lower 4 bits (Bd) select 1 of 16 4-bit digits in the selected data register. While the 4-bit contents of the selected RAM digit (M) is usually loaded into or from, or exchanged with, the A register (accumulator), it may also be loaded into or from the Q latches or loaded from the L ports. RAM addressing may also be performed directly by the LDD and XAD instructions based upon the 6-bit contents of the operand field of these instructions. The Bd register also serves as a source register for 4-bit data sent directly to the D outputs.

### Internal Logic

The 4-bit A register (accumulator) is the source and destination register for most I/O, arithmetic, logic and data memory access operations. It can also be used to load the Br and Bd portions of the B register, to load and input 4 bits of the 8-bit Q latch data, to input 4 bits of the 8-bit L I/O port data and to perform data exchanges with the SIO register.

A 4-bit adder performs the arithmetic and logic functions of the COP420/421L, storing its results in A. It also outputs a carry bit to the 1-bit C register, most often employed to indicate arithmetic overflow. The C register, in conjunction with the XAS instruction and the EN register,

also serves to control the SK output. C can be outputted directly to SK or can enable SK to be a sync clock each instruction cycle time. (See XAS instruction and EN register description, below.)

Four general-purpose inputs, IN<sub>3</sub>-IN<sub>0</sub>, are provided.

The D register provides 4 general-purpose outputs and is used as the destination register for the 4-bit contents of Bd. The D outputs can be directly connected to the digits of a multiplexed LED display.

The G register contents are outputs to 4 general-purpose bidirectional I/O ports. G I/O ports can be directly connected to the digits of a multiplexed LED display.

The Q register is an internal, latched, 8-bit register, used to hold data loaded to or from M and A, as well as 8-bit data from ROM. Its contents are outputted to the L I/O ports when the L drivers are enabled under program control. (See LEI instruction.)

The 8 L drivers, when enabled, output the contents of latched Q data to the L I/O ports. Also, the contents of L may be read directly into A and M. L I/O ports can be directly connected to the segments of a multiplexed LED display (using the LED Direct Drive output configuration option) with Q data being outputted to the Sa-Sg and decimal point segments of the display.

The SIO register functions as a 4-bit serial-in/serial-out shift register or as a binary counter depending on the contents of the EN register. (See EN register description, below.) Its contents can be exchanged with A, allowing it to input or output a continuous serial data stream. SIO may also be used to provide additional parallel I/O by connecting SO to external serial-in/parallel-out shift registers. For example of additional parallel output capacity see Application #2.

The XAS instruction copies C into the SKL latch. In the counter mode, SK is the output of SKL; in the shift register mode, SK outputs SKL ANDed with the clock.

The EN register is an internal 4-bit register loaded under program control by the LEI instruction. The state of each bit of this register selects or deselects the particular feature associated with each bit of the EN register (EN<sub>3</sub>-EN<sub>0</sub>).

1. The least significant bit of the enable register, EN<sub>0</sub>, selects the SIO register as either a 4-bit shift register or a 4-bit binary counter. With EN<sub>0</sub> set, SIO is an asynchronous binary counter, *decrementing* its value by one upon each low-going pulse ("1" to "0") occurring on the SI input. Each pulse must be at least two instruction cycles wide. SK outputs the value of SKL. The SO output is equal to the value of EN<sub>3</sub>. With EN<sub>0</sub> reset, SIO is a serial shift register shifting left each instruction cycle time. The data present at SI goes into the least significant bit of SIO. SO can be enabled to output the most significant bit of SIO each cycle time. (See 4 below.) The SK output becomes a logic-controlled clock.
2. With EN<sub>1</sub> set the IN<sub>1</sub> input is enabled as an interrupt input. Immediately following an interrupt, EN<sub>1</sub> is reset to disable further interrupts.
3. With EN<sub>2</sub> set, the L drivers are enabled to output the data in Q to the L I/O ports. Resetting EN<sub>2</sub> disables



the L drivers, placing the L I/O ports in a high-impedance input state.

4. EN<sub>3</sub>, in conjunction with EN<sub>0</sub>, affects the SO output. With EN<sub>0</sub> set (binary counter option selected) SO will output the value loaded into EN<sub>3</sub>. With EN<sub>0</sub> reset (serial shift register option selected), setting EN<sub>3</sub> enables SO as the output of the SIO shift register,

outputting serial shifted data each instruction time. Resetting EN<sub>3</sub> with the serial shift register option selected disables SO as the shift register output; data continues to be shifted through SIO and can be exchanged with A via an XAS instruction but SO remains reset to "0." The table below provides a summary of the modes associated with EN<sub>3</sub> and EN<sub>0</sub>.

Enable Register Modes — Bits EN<sub>3</sub> and EN<sub>0</sub>

EN <sub>3</sub>	EN <sub>0</sub>	SIO	SI	SO	SK
0	0	Shift Register	Input to Shift Register	0	If SKL = 1, SK = Clock If SKL = 0, SK = 0
1	0	Shift Register	Input to Shift Register	Serial Out	If SKL = 1, SK = Clock If SKL = 0, SK = 0
0	1	Binary Counter	Input to Binary Counter	0	If SKL = 1, SK = 1 If SKL = 0, SK = 0
1	1	Binary Counter	Input to Binary Counter	1	If SKL = 1, SK = 1 If SKL = 0, SK = 0

### Interrupt

The following features are associated with the IN<sub>1</sub> interrupt procedure and protocol and must be considered by the programmer when utilizing interrupts.

- The interrupt, once acknowledged as explained below, pushes the next sequential program counter address (PC + 1) onto the stack, pushing in turn the contents of the other subroutine-save registers to the next lower level (PC + 1 → SA → SB → SC). Any previous contents of SC are lost. The program counter is set to hex address 0FF (the last word of page 3) and EN<sub>1</sub> is reset.
- An interrupt will be acknowledged only after the following conditions are met:
  - EN<sub>1</sub> has been set.
  - A low-going pulse ("1" to "0") at least two instruction cycles wide occurs on the IN<sub>1</sub> input.
  - A currently executing instruction has been completed.
  - All successive transfer of control instructions and successive LBIs have been completed (e.g., if the main program is executing a JP instruction which transfers program control to another JP instruction, the interrupt will not be acknowledged until the second JP instruction has been executed).
- Upon acknowledgement of an interrupt, the skip logic status is saved and later restored upon popping of the stack. For example, if an interrupt occurs during the execution of ASC (Add with Carry, Skip on Carry) instruction which results in carry, the skip logic status is saved and program control is transferred to the interrupt servicing routine at hex address 0FF. At the end of the interrupt routine, a RET instruction is executed to "pop" the stack and return program control to the instruction following the original ASC. At this time, the skip logic is enabled and skips this instruction because of the previous ASC carry. Subroutines and LQID instructions should not be nested

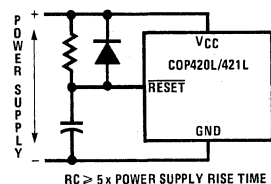
within the interrupt servicing routine since their popping the stack will enable any previously saved main program skips, interfering with the orderly execution of the interrupt routine.

- The first instruction of the interrupt routine at hex address 0FF must be a NOP.
- A LEI instruction can be put immediately before the RET to re-enable interrupts.

### Initialization

The Reset Logic will initialize (clear) the device upon power-up if the power supply rise time is less than 1 ms and greater than 1 μs. If the power supply rise time is greater than 1 ms, the user must provide an external RC network and diode to the RESET pin as shown below. The RESET pin is configured as a Schmitt trigger input. If not used it should be connected to V<sub>CC</sub>. Initialization will occur whenever a logic "0" is applied to the RESET input, provided it stays low for at least three instruction cycle times.

Upon initialization, the PC register is cleared to 0 (ROM address 0) and the A, B, C, D, EN, and G registers are cleared. The SK output is enabled as a SYNC output, providing a pulse each instruction cycle time. Data Memory (RAM) is not cleared upon initialization. The first instruction at address 0 must be a CLRA.



Power-Up Clear Circuit

### Oscillator

There are four basic clock oscillator configurations available as shown by Figure 4.

- a. Crystal Controlled Oscillator.** CKI and CKO are connected to an external crystal. The instruction cycle time equals the crystal frequency divided by 32 (optional by 16 or 8).
- b. External Oscillator.** CKI is an external clock input signal. The external frequency is divided by 32 (optional by 16 or 8) to give the instruction cycle time. CKO is now available to be used as the RAM power supply ( $V_R$ ), as a general purpose input, or as a SYNC input.
- c. RC Controlled Oscillator.** CKI is configured as a single pin RC controlled Schmitt trigger oscillator. The instruction cycle time equals the oscillation frequency divided by 4. CKO is available as the RAM power supply ( $V_R$ ) or as a general purpose input.
- d. Externally Synchronized Oscillator.** Intended for use in multi-COP systems, CKO is programmed to function as an input connected to the SK output of another COP chip operating at the same frequency (COP chip with L or C suffix) with CKI connected as shown. In this configuration, the SK output connected to CKO must provide a SYNC (instruction cycle) signal to CKO, thereby allowing synchronous data transfer between the COPs using only the SI and SO serial I/O pins in conjunction with the XAS instruction. Note that on power-up SK is automatically enabled as a

SYNC output (See Functional Description, Initialization, above).

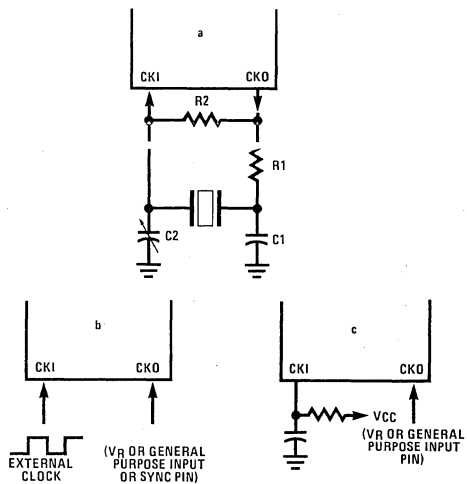
### CKO Pin Options

In a crystal controlled oscillator system, CKO is used as an output to the crystal network. As an option CKO can be a SYNC input as described above. As another option CKO can be a general purpose input, read into bit 2 of A (accumulator) upon execution of an INIL instruction. As another option, CKO can be a RAM power supply pin ( $V_R$ ), allowing its connection to a standby/backup power supply to maintain the integrity of RAM data with minimum power drain when the main supply is inoperative or shut down to conserve power. Using either option is appropriate in applications where the COP420L/421L system timing configuration does not require use of the CKO pin.

### RAM Keep-Alive Option (Not available on COP422L)

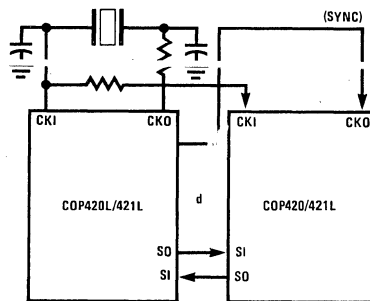
Selecting CKO as the RAM power supply ( $V_R$ ) allows the user to shut off the chip power supply ( $V_{CC}$ ) and maintain data in the RAM. To insure that RAM data integrity is maintained, the following conditions must be met:

1.  $\overline{RESET}$  must go low before  $V_{CC}$  goes below spec during power-off;  $V_{CC}$  must be within spec before  $\overline{RESET}$  goes high on power-up.
2. During normal operation  $V_R$  must be within the operating range of the chip, with  $(V_{CC} - 1) \leq V_R \leq V_{CC}$ .
3.  $V_R$  must be  $\geq 3.3V$  with  $V_{CC}$  off.



Crystal Oscillator

Crystal Value	Component Values			
	R1 ( $\Omega$ )	R2 ( $\Omega$ )	C1 (pF)	C2 (pF)
455 kHz	4.7k	1M	220	220
2.097 MHz	1k	1M	30	6-36



RC Controlled Oscillator

R (k $\Omega$ )	C (pF)	Instruction Cycle Time ( $\mu$ s)
51	100	19 $\pm$ 15%
82	56	19 $\pm$ 13%

Note: 200k  $\geq$  R  $\geq$  25k  
360 pF  $\geq$  C  $\geq$  50 pF

Figure 4. COP420/421L Oscillator

## I/O Options

COP420L/421L outputs have the following optional configurations, illustrated in Figure 5:

- a. **Standard** — an enhancement-mode device to ground in conjunction with a depletion-mode device to  $V_{CC}$ , compatible with LSTTL and CMOS input requirements. Available on SO, SK, and all D and G outputs.
- b. **Open-Drain** — an enhancement-mode device to ground only, allowing external pull-up as required by the user's application. Available on SO, SK, and all D and G outputs.
- c. **Push-Pull** — An enhancement-mode device to ground in conjunction with a depletion-mode device paralleled by an enhancement-mode device to  $V_{CC}$ . This configuration has been provided to allow for fast rise and fall times when driving capacitive loads. Available on SO and SK outputs only.
- d. **Standard L** — same as a., but may be disabled. Available on L outputs only.
- e. **Open Drain L** — same as b., but may be disabled. Available on L outputs only.
- f. **LED Direct Drive** — an enhancement-mode device to ground and to  $V_{CC}$ , meeting the typical current sourcing requirements of the segments of an LED display. The sourcing device is clamped to limit current flow. These devices may be turned off under program control (See Functional Description, EN Register), placing the outputs in a high-impedance state to provide required LED segment blanking for a multiplexed display. Available on L outputs only.
- g. **TRI-STATE® Push-Pull** — an enhancement-mode device to ground and  $V_{CC}$ . These outputs are TRI-STATE outputs, allowing for connection of these outputs to a data bus shared by other bus drivers. Available on L outputs only.

COP420L/COP421L inputs have the following optional configurations:

- h. An on-chip depletion load device to  $V_{CC}$ .
- i. A Hi-Z input which must be driven to a "1" or "0" by external components.

The above input and output configurations share common enhancement-mode and depletion-mode devices. Specifically, all configurations use one or more of six devices (numbered 1-6, respectively). Minimum and maximum current ( $I_{OUT}$  and  $V_{OUT}$ ) curves are given in Figure 6 for each of these devices to allow the designer to effectively use these I/O configurations in designing a COP420L/421L system.

The SO, SK outputs can be configured as shown in a., b., or c. The D and G outputs can be configured as shown in a. or b. Note that when inputting data to the G ports, the G outputs should be set to "1". The L outputs can be configured as in d., e., f. or g.

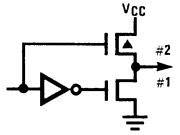
An important point to remember if using configuration d. or f. with the L drivers is that even when the L drivers are disabled, the depletion load device will source a small amount of current (see Figure 6, device 2); however, when the L lines are used as inputs, the disabled depletion device *cannot* be relied on to source sufficient current to pull an input to a logic 1.

## COP421L

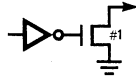
If the COP420L is bonded as a 24-pin device, it becomes the COP421L, illustrated in Figure 2, COP420L/421L Connection Diagrams. Note that the COP421L does not contain the four general purpose IN inputs ( $IN_3$ - $IN_0$ ). Use of this option precludes, of course, use of the IN options and the interrupt feature. All other options are available for the COP421L.

## COP422L

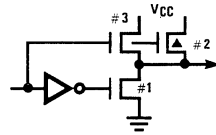
If the COP421L is bonded as a 20-pin device, it becomes the COP422L, as illustrated in Figure 2. Note that the COP422L contains all the COP421L pins except  $D_0$ ,  $D_1$ ,  $G_0$ , and  $G_1$ . COP422L also does not allow RAM power supply input as a valid CKO pin option.



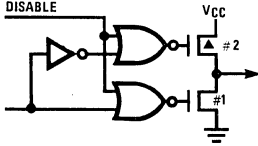
a. Standard Output



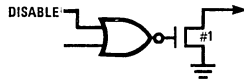
b. Open-Drain Output



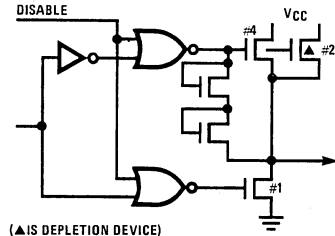
c. Push-Pull Output



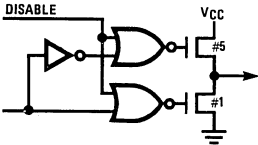
d. Standard L Output



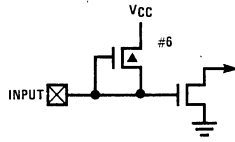
e. Open-Drain L Output



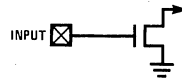
f. LED (L Output)



g. TRI-STATE® Push-Pull (L Output)



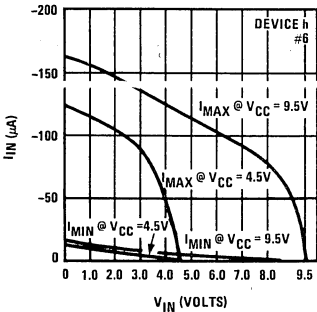
h. Input with Load



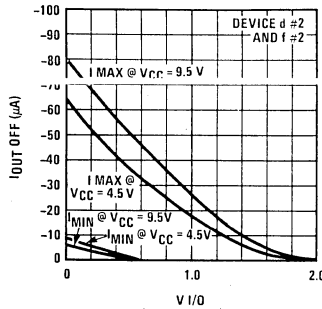
i. Hi-Z Input

Figure 5. Output Configurations

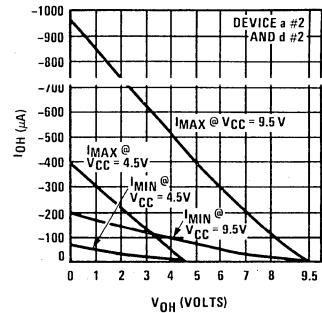
Input current  $I_{IN_0}$ - $I_{IN_3}$



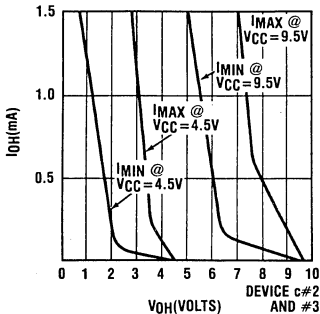
Input current for  $L_0$ - $L_7$  when output programmed OFF by software



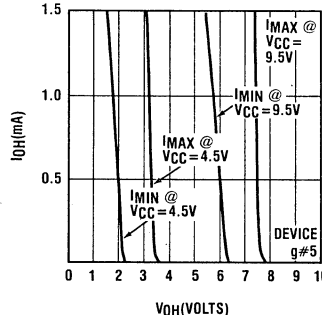
Source current for standard output configuration



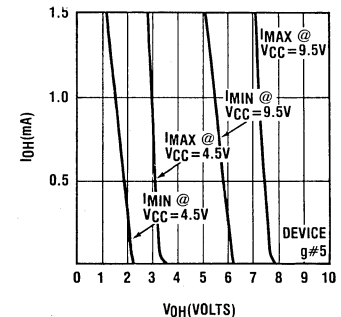
Source current for SO and SK in push-pull configuration



Source Current for  $L_0$ - $L_7$  in TRI-STATE® Configuration (High Current Option)



Source Current for  $L_0$ - $L_7$  in TRI-STATE® Configuration (Low Current Option)



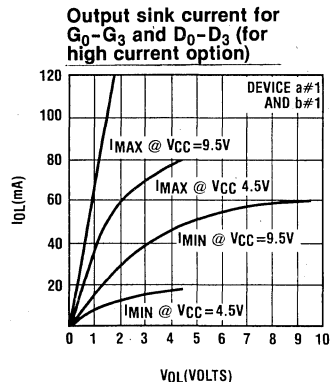
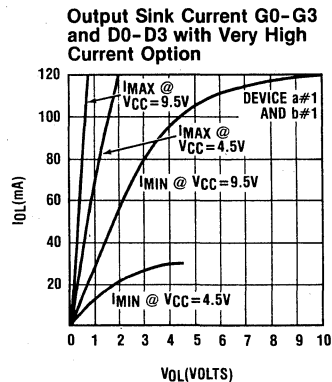
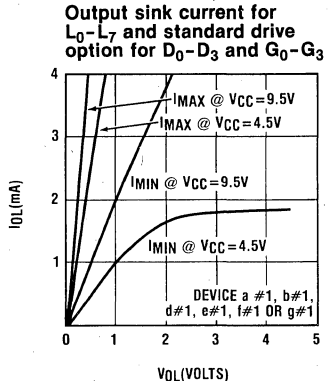
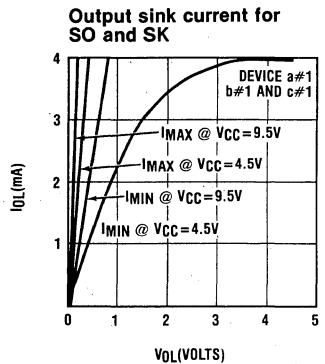
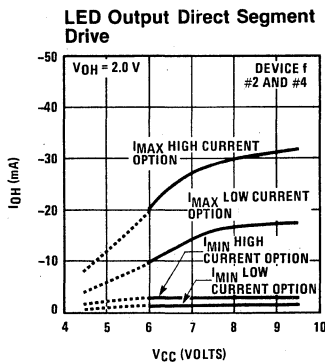
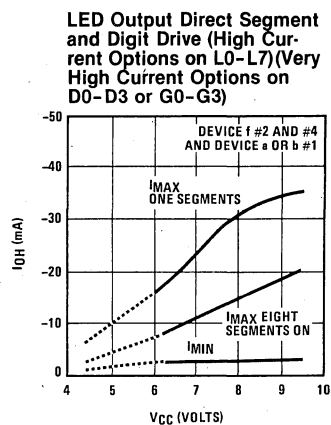
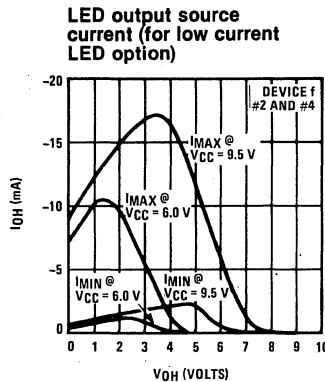
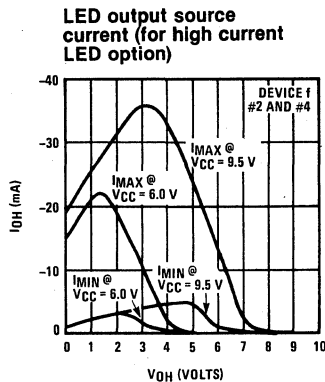


Figure 6. COP420/COP421L/COP422L Input/Output Characteristics

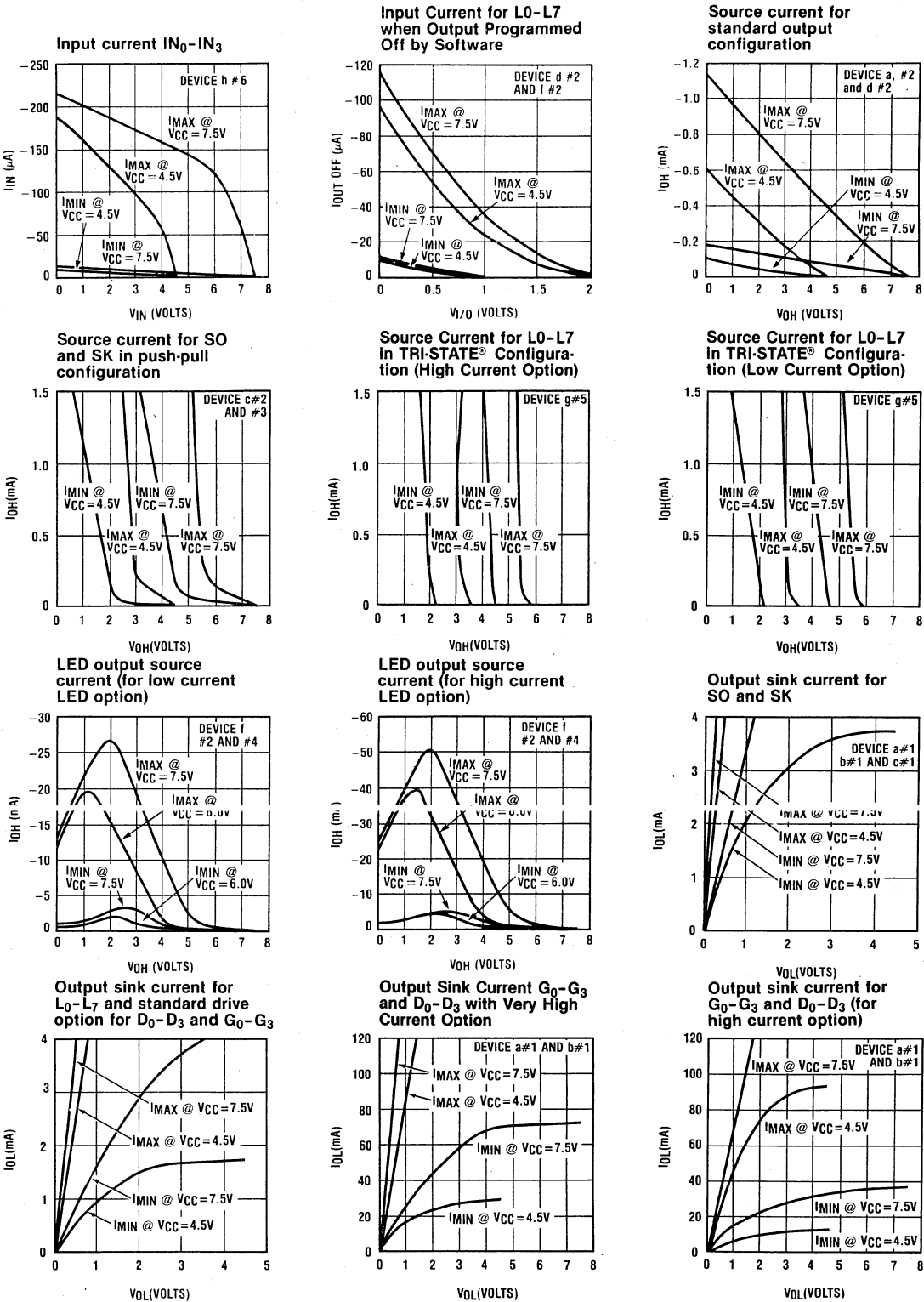


Figure 7. COP320L/COP321L/COP322L Input/Output Characteristics



Table 2. COP420L/421L Instruction Set

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>ARITHMETIC INSTRUCTIONS</b>						
ASC		30	$\boxed{0011 0000}$	$A + C + \text{RAM}(B) \rightarrow A$ Carry $\rightarrow C$	Carry	Add with Carry, Skip on Carry
ADD		31	$\boxed{0011 0001}$	$A + \text{RAM}(B) \rightarrow A$	None	Add RAM to A
ADT		4A	$\boxed{0100 1010}$	$A + 10_{10} \rightarrow A$	None	Add Ten to A
AISC	y	5-	$\boxed{0101 y}$	$A + y \rightarrow A$	Carry	Add Immediate, Skip on Carry ( $y \neq 0$ )
CASC		10	$\boxed{0001 0000}$	$\bar{A} + \text{RAM}(B) + C \rightarrow A$ Carry $\rightarrow C$	Carry	Complement and Add with Carry, Skip on Carry
CLRA		00	$\boxed{0000 0000}$	$0 \rightarrow A$	None	Clear A
COMP		40	$\boxed{0100 0000}$	$\bar{A} \rightarrow A$	None	Ones complement of A to A
NOP		44	$\boxed{0100 0100}$	None	None	No Operation
RC		32	$\boxed{0011 0010}$	"0" $\rightarrow C$	None	Reset C
SC		22	$\boxed{0010 0010}$	"1" $\rightarrow C$	None	Set C
XOR		02	$\boxed{0000 0010}$	$A \oplus \text{RAM}(B) \rightarrow A$	None	Exclusive-OR RAM with A
<b>TRANSFER OF CONTROL INSTRUCTIONS</b>						
JID		FF	$\boxed{1111 1111}$	$\text{ROM}(\text{PC}_{9:8}, A, M) \rightarrow \text{PC}_{7:0}$	None	Jump Indirect (Note 3)
JMP	a	6-	$\boxed{0110 00 a_{9:8}}$ $\boxed{a_{7:0}}$	$a \rightarrow \text{PC}$	None	Jump
JP	a	--	$\boxed{1 a_{6:0}}$ (pages 2,3 only) or $\boxed{11 a_{5:n}}$ (all other pages)	$a \rightarrow \text{PC}_{6:0}$  $a \rightarrow \text{PC}_{5:n}$	None	Jump within Page (Note 4)
JSRP	a	--	$\boxed{10 a_{5:0}}$	$\text{PC} + 1 \rightarrow \text{SA} \rightarrow \text{SB} \rightarrow \text{SC}$ $0010 \rightarrow \text{PC}_{9:6}$ $a \rightarrow \text{PC}_{5:0}$	None	Jump to Subroutine Page (Note 5)
JSR	a	6-	$\boxed{0110 10 a_{9:8}}$ $\boxed{a_{7:0}}$	$\text{PC} + 1 \rightarrow \text{SA} \rightarrow \text{SB} \rightarrow \text{SC}$ $a \rightarrow \text{PC}$	None	Jump to Subroutine
RET		48	$\boxed{0100 1000}$	$\text{SC} \rightarrow \text{SB} \rightarrow \text{SA} \rightarrow \text{PC}$	None	Return from Subroutine
RETSK		49	$\boxed{0100 1001}$	$\text{SC} \rightarrow \text{SB} \rightarrow \text{SA} \rightarrow \text{PC}$	Always Skip on Return	Return from Subroutine then Skip



Table 2. COP420L/421L Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description								
MEMORY REFERENCE INSTRUCTIONS														
CAMQ		33	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	1	1	0	0	1	1	A → Q7:4 RAM(B) → Q3:0	None	Copy A, RAM to Q
	0	0	1	1	0	0	1	1						
	3C	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr></table>	0	0	1	1	1	1	0	0				
0	0	1	1	1	1	0	0							
CQMA		33	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	1	1	0	0	1	1	Q7:4 → RAM(B) Q3:0 → A	None	Copy Q to RAM, A
	0	0	1	1	0	0	1	1						
	2C	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr></table>	0	0	1	0	1	1	0	0				
0	0	1	0	1	1	0	0							
LD	r	-5	<table border="1"><tr><td>0</td><td>0</td><td>r</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td></tr></table>	0	0	r	0	1	0	1	0	RAM(B) → A Br ⊕ r → Br	None	Load RAM into A, Exclusive-OR Br with r
0	0	r	0	1	0	1	0							
LDD	r,d	23	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	1	0	0	0	0	1	RAM(r,d) → A	None	Load A with RAM pointed to directly by r,d
		0	0	1	0	0	0	0	1					
--	<table border="1"><tr><td>0</td><td>0</td><td>r</td><td>d</td><td></td><td></td><td></td><td></td></tr></table>	0	0	r	d									
0	0	r	d											
LQID		BF	<table border="1"><tr><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr></table>	1	0	1	1	1	1	1	1	ROM(PC9:8,A,M) → Q SB → SC	None	Load Q Indirect (Note 3)
1	0	1	1	1	1	1	1							
RMB	0	4C	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr></table>	0	1	0	0	1	1	0	0	0 → RAM(B) <sub>0</sub>	None	Reset RAM Bit
	0	1	0	0	1	1	0	0						
	1	45	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr></table>	0	1	0	0	0	1	0	1	0 → RAM(B) <sub>1</sub>		
	0	1	0	0	0	1	0	1						
2	42	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table>	0	1	0	0	0	0	1	0	0 → RAM(B) <sub>2</sub>			
0	1	0	0	0	0	1	0							
3	43	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	1	0	0	0	0	1	1	0 → RAM(B) <sub>3</sub>			
0	1	0	0	0	0	1	1							
SMB	0	4D	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr></table>	0	1	0	0	1	1	0	1	1 → RAM(B) <sub>0</sub>	None	Set RAM Bit
	0	1	0	0	1	1	0	1						
	1	47	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr></table>	0	1	0	0	1	1	0	1	1 → RAM(B) <sub>1</sub>		
	0	1	0	0	1	1	0	1						
2	46	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td></tr></table>	0	1	0	0	0	1	1	0	1 → RAM(B) <sub>2</sub>			
0	1	0	0	0	1	1	0							
3	4B	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr></table>	0	1	0	0	1	0	1	1	1 → RAM(B) <sub>3</sub>			
0	1	0	0	1	0	1	1							
STII	y	7-	<table border="1"><tr><td>0</td><td>1</td><td>1</td><td>1</td><td>y</td><td></td><td></td><td></td></tr></table>	0	1	1	1	y				y → RAM(B) Bd + 1 → Bd	None	Store Memory Immediate and Increment Bd
0	1	1	1	y										
X	r	-6	<table border="1"><tr><td>0</td><td>0</td><td>r</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr></table>	0	0	r	0	1	1	1	0	RAM(B) ↔ A Br ⊕ r → Br	None	Exchange RAM with A, Exclusive-OR Br with r
0	0	r	0	1	1	1	0							
XAD	r,d	23	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	1	0	0	0	1	1	RAM(r,d) ↔ A	None	Exchange A with RAM pointed to directly by r,d
		0	0	1	0	0	0	1	1					
--	<table border="1"><tr><td>1</td><td>0</td><td>r</td><td>d</td><td></td><td></td><td></td><td></td></tr></table>	1	0	r	d									
1	0	r	d											
XDS	r	-7	<table border="1"><tr><td>0</td><td>0</td><td>r</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td></tr></table>	0	0	r	0	1	1	1	1	RAM(B) ↔ A Bd - 1 → Bd Br ⊕ r → Br	Bd decrements past 0	Exchange RAM with A and Decrement Bd, Exclusive-OR Br with r
0	0	r	0	1	1	1	1							
XIS	r	-4	<table border="1"><tr><td>0</td><td>0</td><td>r</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	r	0	1	0	0	0	RAM(B) ↔ A Bd + 1 → Bd Br ⊕ r → Br	Bd increments past 15	Exchange RAM with A and Increment Bd, Exclusive-OR Br with r
0	0	r	0	1	0	0	0							
REGISTER REFERENCE INSTRUCTIONS														
CAB		50	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	1	0	1	0	0	0	0	A → Bd	None	Copy A to Bd
0	1	0	1	0	0	0	0							
CBA		4E	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td></tr></table>	0	1	0	0	1	1	1	1	Bd → A	None	Copy Bd to A
0	1	0	0	1	1	1	1							
LBI	r,d	--	<table border="1"><tr><td>0</td><td>0</td><td>r</td><td>(d-1)</td><td></td><td></td><td></td><td></td></tr></table> (d = 0, 9:15)	0	0	r	(d-1)					r,d → B	Skip until not a LBI	Load B Immediate with r,d (Note 6)
		0	0	r	(d-1)									
		33	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	1	1	0	0	1	1			
0	0	1	1	0	0	1	1							
--	<table border="1"><tr><td>1</td><td>0</td><td>r</td><td>d</td><td></td><td></td><td></td><td></td></tr></table> (any d)	1	0	r	d									
1	0	r	d											
LEI	y	33	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	1	1	0	0	1	1	y → EN	None	Load EN Immediate (Note 7)
		0	0	1	1	0	0	1	1					
6-	<table border="1"><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>y</td><td></td><td></td><td></td></tr></table>	0	1	1	0	y								
0	1	1	0	y										
XABR		12	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table>	0	0	0	1	0	0	1	0	A ↔ Br (0,0 → A <sub>3</sub> ,A <sub>2</sub> )	None	Exchange A with Br
0	0	0	1	0	0	1	0							

Table 2. COP420L/421L Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>TEST INSTRUCTIONS</b>						
SKC		20	00100000		C = "1"	Skip if C is True
SKE		21	00100001		A = RAM(B)	Skip if A Equals RAM
SKGZ		33	00110011		G <sub>3:0</sub> = 0	Skip if G is Zero (all 4 bits)
SKGBZ		21	00100001			
		33	00110011	1st byte		Skip if G Bit is Zero
	0	01	00000001	} 2nd byte	G <sub>0</sub> = 0	
	1	11	00010001		G <sub>1</sub> = 0	
	2	03	00000011		G <sub>2</sub> = 0	
3	13	00010011	G <sub>3</sub> = 0			
SKMBZ	0	01	00000001		RAM(B) <sub>0</sub> = 0	Skip if RAM Bit is Zero
	1	11	00010001		RAM(B) <sub>1</sub> = 0	
	2	03	00000011		RAM(B) <sub>2</sub> = 0	
	3	13	00010011		RAM(B) <sub>3</sub> = 0	
SKT		41	01000001		A time-base counter carry has occurred since last test	Skip on Timer (Note 3)
<b>INPUT/OUTPUT INSTRUCTIONS</b>						
ING		33	00110011	G → A	None	Input G Ports to A
		2A	00101010			
ININ		33	00110011	IN → A	None	Input IN Inputs to A (Note 2)
		28	00101000			
INIL		33	00110011	IL <sub>3</sub> , CKO, "0", IL <sub>0</sub> → A	None	Input IL Latches to A (Note 3)
		29	00101001			
INL		33	00110011	L <sub>7:4</sub> → RAM(B)	None	Input L Ports to RAM,A
		22	00101010	L <sub>3:0</sub> → A		
OBD		33	00110011	Bd → D	None	Output Bd to D Outputs
		3E	00111110			
OGI	y	33	00110011	y → G	None	Output to G Ports Immediate
	5-		0101 y			
OMG		33	00110011	RAM(B) → G	None	Output RAM to G Ports
		3A	00111010			
XAS		4F	01001111	A ↔ SIO, C → SKL	None	Exchange A with SIO (Note 3)

**Note 1:** All subscripts for alphabetical symbols indicate bit numbers unless explicitly defined (e.g., Br and Bd are explicitly defined). Bits are numbered 0 to N where 0 signifies the least significant bit (low-order, right-most bit). For example, A<sub>3</sub> indicates the most significant (left-most) bit of the 4-bit A register.

**Note 2:** The ININ instruction is only available on the 28-pin COP420L as the other devices do not contain the IN inputs.

**Note 3:** For additional information on the operation of the XAS, JID, LQID, INIL, and SKT instructions, see below.

**Note 4:** The JP instruction allows a jump, while in subroutine pages 2 or 3, to any ROM location within the two-page boundary of pages 2 or 3. The JP instruction, otherwise, permits a jump to a ROM location within the current 64-word page. JP may not jump to the last word of a page.

**Note 5:** A JSRP transfers program control to subroutine page 2 (0010 is loaded into the upper 4 bits of P). A JSRP may not be used when in pages 2 or 3. JSRP may not jump to the last word in page 2.

**Note 6:** LBI is a single-byte instruction if d = 0, 9, 10, 11, 12, 13, 14, or 15. The machine code for the lower 4 bits equals the binary value of the "d" data minus 7, e.g., to load the lower four bits of B (Bd) with the value 9 (1001<sub>2</sub>), the lower 4 bits of the LBI instruction equal 8 (1000<sub>2</sub>). To load 0, the lower 4 bits of the LBI instruction should equal 15 (1111<sub>2</sub>).

**Note 7:** Machine code for operand field y for LEI instruction should equal the binary value to be latched into EN, where a "1" or "0" in each bit of EN corresponds with the selection or deselection of a particular function associated with each bit. (See Functional Description, EN Register.)

The following information is provided to assist the user in understanding the operation of several unique instructions and to provide notes useful to programmers in writing COP420L/421L programs.

### XAS Instruction

XAS (Exchange A with SIO) exchanges the 4-bit contents of the accumulator with the 4-bit contents of the SIO register. The contents of SIO will contain serial-in/serial-out shift register or binary counter data, depending on the value of the EN register. An XAS instruction will also affect the SK output. (See Functional Description, EN Register, above.) If SIO is selected as a shift register, an XAS instruction must be performed once every 4 instruction cycles to effect a continuous data stream.

### JID Instruction

JID (Jump Indirect) is an indirect addressing instruction, transferring program control to a new ROM location pointed to indirectly by A and M. It loads the lower 8 bits of the ROM address register PC with the contents of ROM addressed by the 10-bit word, PC<sub>9:8</sub>, A, M. PC<sub>9</sub> and PC<sub>8</sub> are not affected by this instruction.

Note that JID requires 2 instruction cycles to execute.

### INIL Instruction

INIL (Input IL Latches to A) inputs 2 latches, IL<sub>3</sub> and IL<sub>0</sub> (see Figure 8) and CKO into A. The IL<sub>3</sub> and IL<sub>0</sub> latches are set if a low-going pulse ("1" to "0") has occurred on the IN<sub>3</sub> and IN<sub>0</sub> inputs since the last INIL instruction, provided the input pulse stays low for at least two instruction times. Execution of an INIL inputs IL<sub>3</sub> and IL<sub>0</sub> into A3 and A0 respectively, and resets these latches to allow them to respond to subsequent low-going pulses on the IN<sub>3</sub> and IN<sub>0</sub> lines. If CKO is mask programmed as a general purpose input, an INIL will input the state of CKO into A2. If CKO has not been so programmed, a "1" will be placed in A2. A "0" is always placed in A1 upon the execution of an INIL. The general purpose inputs IN<sub>3</sub>-IN<sub>0</sub> are input to A upon execution of an ININ instruction. (See Table 2, ININ instruction.) INIL is useful in recognizing pulses of short duration or pulses which occur too often to be read conveniently by an ININ instruction. IL latches are *not cleared* on reset.

### LQID Instruction

LQID (Load Q Indirect) loads the 8-bit Q register with the contents of ROM pointed to by the 10-bit word PC<sub>9</sub>, PC<sub>8</sub>, A, M. LQID can be used for table lookup or code conversion such as BCD to seven-segment. The LQID instruction "pushes" the stack (PC + 1 → SA → SB → SC) and replaces the least significant 8 bits of PC as follows: A → PC<sub>7:4</sub>, RAM(B) → PC<sub>3:0</sub>, leaving PC<sub>9</sub> and PC<sub>8</sub> unchanged. The ROM data pointed to by the new address is fetched and loaded into the Q latches. Next, the stack is "popped" (SC → SB → SA → PC), restoring the saved value of PC to continue sequential program execution. Since LQID pushes SB → SC, the previous contents of SC are lost. Also, when LQID pops the stack, the previously pushed contents of SB are left in SC. The net result is that the contents of SB are placed in SC (SB → SC). Note that LQID takes two instruction cycle times to execute.

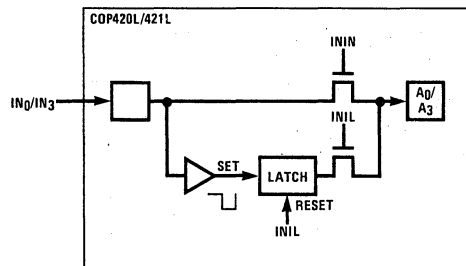


Figure 8. INIL Hardware Implementation

### SKT Instruction

The SKT (Skip On Timer) instruction tests the state of an internal 10-bit time-base counter. This counter divides the instruction cycle clock frequency by 1024 and provides a latched indication of counter overflow. The SKT instruction tests this latch, executing the next program instruction if the latch is not set. If the latch has been set since the previous test, the next program instruction is skipped and the latch is reset. The features associated with this instruction, therefore, allow the COP420L/421L to generate its own time-base for real-time processing rather than relying on an external input signal.

For example, using a 2.097 MHz crystal as the time-base to the clock generator, the instruction cycle clock frequency will be 65kHz (crystal frequency ÷ 32) and the binary counter output pulse frequency will be 64Hz. For time-of-day or similar real-time processing, the SKT instruction can call a routine which increments a "seconds" counter every 64 ticks.

### Instruction Set Notes

- The first word of a COP420L/421L program (ROM address 0) must be a CLRA (Clear A) instruction.
- Although skipped instructions are not executed, one instruction cycle time is devoted to skipping each byte of the skipped instruction. Thus all program paths except JID and LQID take the same number of cycle times whether instructions are skipped or executed. JID and LQID instructions take 2 cycles if executed and 1 cycle if skipped.
- The ROM is organized into 16 pages of 64 words each. The Program Counter is a 10-bit binary counter, and will count through page boundaries. If a JP, JSRP, JID or LQID instruction is located in the last word of a page, the instruction operates as if it were in the next page. For example: a JP located in the last word of a page will jump to a location in the next page. Also, a LQID or JID located in the last word of page 3, 7, 11, or 15 will access data in the next group of four pages.

## Option List

The COP420L/421L mask-programmable options are assigned numbers which correspond with the COP420L pins.

The following is a list of COP420L options. When specifying a COP421L chip, Options 9, 10, 19, and 20 must all be set to zero. When specifying a COP422L chip, options 9, 10, 19, and 20 must all be set to zero; options 21 and 22 may not be set to one, three or five; and option 2 may not be set to one. The options are programmed at the same time as the ROM pattern to provide the user with the hardware flexibility to interface to various I/O components using little or no external circuitry.

- Option 1 = 0: Ground Pin — no options available
- Option 2: CKO Output  
 = 0: clock generator output to crystal/resonator (0 not allowable value if Option 3 = 3)  
 = 1: pin is RAM power supply ( $V_R$ ) input (not available on the COP422L)  
 = 2: general purpose input with load device to  $V_{CC}$   
 = 3: general purpose input, Hi-Z  
 = 4: multi-COP SYNC input (CKI + 32, CKI + 16)  
 = 5: multi-COP SYNC input (CKI + 8)
- Option 3: CKI Input  
 = 0: oscillator input divided by 32 (2MHz max.)  
 = 1: oscillator input divided by 16 (1MHz max.)  
 = 2: oscillator input divided by 8 (500kHz max.)  
 = 3: single-pin RC controlled oscillator (+4)  
 = 4: Schmitt trigger clock input (+4)
- Option 4: RESET Input  
 = 0: load device to  $V_{CC}$   
 = 1: Hi-Z input
- Option 5:  $L_7$  Driver  
 = 0: Standard output  
 = 1: Open-drain output  
 = 2: High current LED direct segment drive output  
 = 3: High current TRI-STATE® push-pull output  
 = 4: Low-current LED direct segment drive output  
 = 5: Low-current TRI-STATE® push-pull output
- Option 6:  $L_6$  Driver  
 same as Option 5
- Option 7:  $L_5$  Driver  
 same as Option 5
- Option 8:  $L_4$  Driver  
 same as Option 5
- Option 9:  $IN_1$  Input  
 = 0: load device to  $V_{CC}$   
 = 1: Hi-Z input
- Option 10:  $IN_2$  Input  
 same as Option 9
- Option 11:  $V_{CC}$  pin  
 = 0: Standard  $V_{CC}$   
 = 1: Optional higher voltage  $V_{CC}$
- Option 12:  $L_3$  Driver  
 same as Option 5
- Option 13:  $L_2$  Driver  
 same as Option 5
- Option 14:  $L_1$  Driver  
 same as Option 5
- Option 15:  $L_0$  Driver  
 same as Option 5
- Option 16: SI Input  
 same as Option 9
- Option 17: SO Driver  
 = 0: standard output  
 = 1: open-drain output  
 = 2: push-pull output
- Option 18: SK Driver  
 same as Option 17
- Option 19:  $IN_0$  Input  
 same as Option 9
- Option 20:  $IN_3$  Input  
 same as Option 9
- Option 21:  $G_0$  I/O Port  
 = 0: very-high current standard output  
 = 1: very-high current open-drain output  
 = 2: high current standard output  
 = 3: high current open-drain output  
 = 4: standard LSTTL output (fanout = 1)  
 = 5: open-drain LSTTL output (fanout = 1)
- Option 22:  $G_1$  I/O Port  
 same as Option 21
- Option 23:  $G_2$  I/O Port  
 same as Option 21
- Option 24:  $G_3$  I/O Port  
 same as Option 21
- Option 25:  $D_3$  Output  
 same as Option 21
- Option 26:  $D_2$  Output  
 same as Option 21
- Option 27:  $D_1$  Output  
 same as Option 21
- Option 28:  $D_0$  Output  
 same as Option 21
- Option 29: L Input Levels  
 = 0: standard TTL input levels ("0" = 0.8V, "1" = 2.0V)  
 = 1: higher voltage input levels ("0" = 1.2V, "1" = 3.6V)
- Option 30:  $IN$  Input Levels  
 same as Option 29
- Option 31:  $G$  Input Levels  
 same as Option 29
- Option 32:  $SI$  Input Levels  
 same as Option 29
- Option 33:  $\overline{RESET}$  Input  
 = 0: Schmitt trigger input  
 = 1: standard TTL input levels  
 = 2: higher voltage input levels
- Option 34: CKO Input Levels (CKO = input; Option 2 = 2,3)  
 same as Option 29
- Option 35: COP Bonding  
 = 0: COP420L (28-pin device)  
 = 1: COP421L (24-pin device)  
 = 2: 28- and 24-pin versions  
 = 3: COP422L (20-pin device)  
 = 4: 28- and 20-pin versions  
 = 5: 24- and 20-pin versions  
 = 6: 28-, 24-, and 20-pin versions

### TEST MODE (Non-Standard Operation)

The SO output has been configured to provide for standard test procedures for the custom-programmed COP420L. With SO forced to logic "1", two test modes are provided, depending upon the value of SI:

- RAM and Internal Logic Test Mode (SI = 1)
- ROM Test Mode (SI = 0)

These special test modes should not be employed by the user; they are intended for manufacturing test only.

### APPLICATION #1: COP420L General Controller

Figure 9 shows an interconnect diagram for a COP420L used as a general controller. Operation of the system is as follows:

- The L<sub>7</sub>-L<sub>0</sub> outputs are configured as LED Direct Drive outputs, allowing direct connection to the segments of the display.

- The D<sub>3</sub>-D<sub>0</sub> outputs drive the digits of the multiplexed display directly and scan the columns of the 4 × 4 keyboard matrix.
- The IN<sub>3</sub>-IN<sub>0</sub> inputs are used to input the 4 rows of the keyboard matrix. Reading the IN lines in conjunction with the current value of the D outputs allows detection, debouncing, and decoding of any one of the 16 keyswitches.
- CKI is configured as a single-pin oscillator input allowing system timing to be controlled by a single-pin RC network. CKO is therefore available for use as a V<sub>R</sub> RAM power supply pin. RAM data integrity is thereby assured when the main power supply is shut down (see RAM Keep-Alive option description).
- SI is selected as the input to a binary counter input. With SIO used as a binary counter, SO and SK can be used as general purpose outputs.
- The 4 bidirectional G I/O ports (G<sub>3</sub>-G<sub>0</sub>) are available for use as required by the user's application.

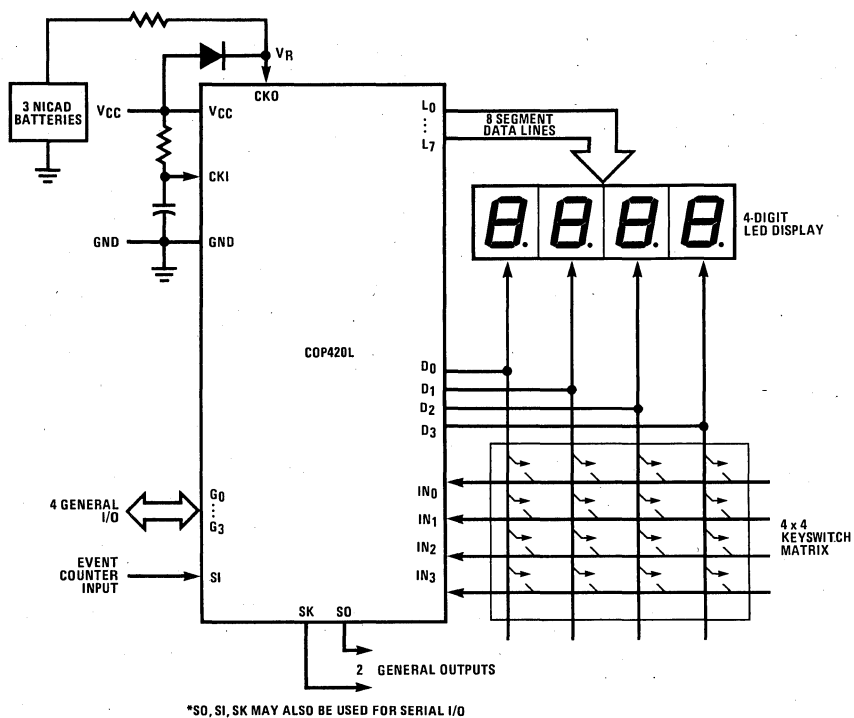
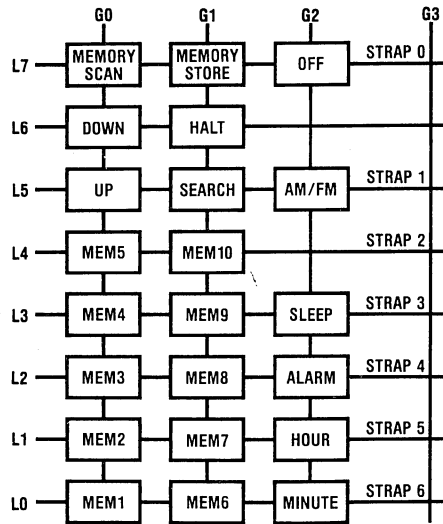


Figure 9. COP420L Keyboard/Display Interface

**APPLICATION #2:**  
Digitally Tuned Radio Controller and Clock



Keyboard Matrix Configuration

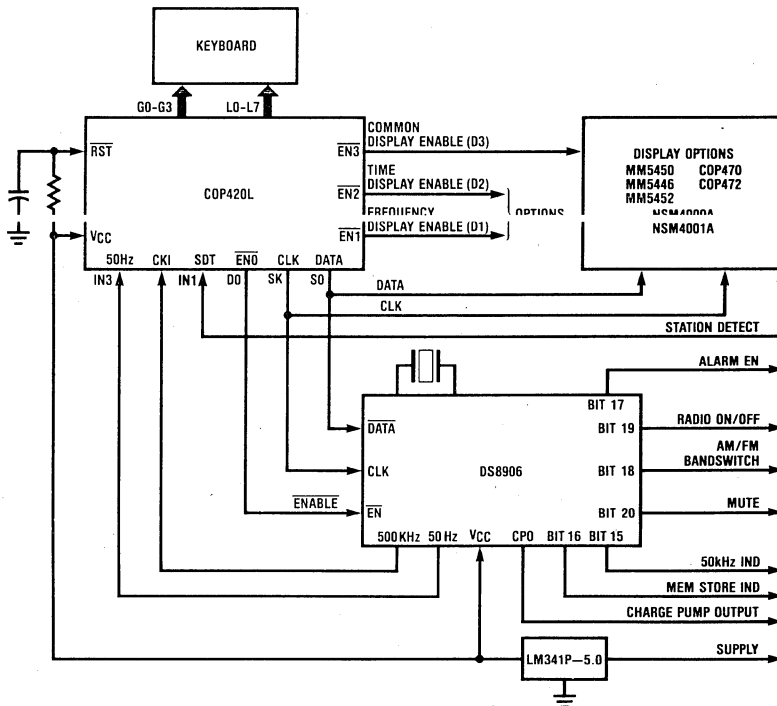


Figure 10. Digital Tuning System Block

## Functional Description

### Logic I/Os

**CKI Input:** This input accepts an external 500kHz signal, divides it by eight and outputs the quotient at the CLK output as the system clock.

**RST Input:** Schmitt trigger input to clear device upon initialization.

**SDT Input:** Interrupt input for station detection. The SDT signal is generated by the radio's station detector and used by the COP420L-HSB to determine if there is a valid station on the active frequency. The status of the SDT input is only relevant during station searching mode. A high on SDT will temporarily terminate the search mode for eight seconds.

**ALM Input:** A high on ALM will activate alarm output via slave device at alarm time. A low on the input will disable alarm function.

**DATA Output:** Push-pull output providing serial data to external devices.

**CLK Output:** Push-pull output providing system clock at data transmitting time.

**50Hz Input:** A normally high input to accept a 50Hz external time base for real-time calculation.

### Momentary Keys Description

**MEM 1-MEM 10:** Each memory represents data of a favorite station in a certain band. Depression of one of these keys will recall the previous stored data and transmit it to the PLL. The PLL will in turn change the radio's receiving frequency as well as the band if necessary. Memory recall keys can also turn on the radio.

**UP:** This key will manually increment receiving frequency. The first four steps of increment will be for fine tuning a station, after which will be fast slewing meant for manual receive frequency changing.

**DOWN:** Has the same function as UP key except that frequency is decremented.

**MEMORY SCAN:** This will start the radio scanning through all ten memories automatically at eight seconds per memory starting from Memory 1. This will also turn on the radio if it was off.

**MEMORY STORE:** Enables the memory store mode which lasts for three seconds. Depression of any memory key will store the active frequency and band in that memory and disable the store mode. Any function key will also disable the mode to prevent memory data being accidentally destroyed.

**HALT:** Depression of the HALT key will stop the search and scan functions at current frequency or memory. HALT also turns on the radio during off time and recall frequency display in single display mode.

**SEARCH:** Activates station searching in the current band. Search speed is 50ms per frequency step with wrapping around at end of band. An 8-second stop will take place on reaching a valid station. The HALT key or any function key will terminate the search. Search direction will normally be upwards unless the DOWN key has been depressed prior to the SEARCH key or during the search function in which case search direction will be downwards.

**OFF:** Turns off the radio or alarm when active.

**AM/FM:** Radio band switch.

**SLEEP:** Activates sleep mode, turns on radio on depression and off radio at the end of sleep period. Setting of sleep period is done by depressing the SLEEP and MINUTE key simultaneously.

**ALARM:** Enables alarm time setting. Depressing the HOUR or MINUTE key and ALARM key simultaneously will set the alarm hour and minute respectively.

**HOUR:** Sets the hour digits of time-related functions.

**MINUTE:** Sets the minute digits of time-related functions.

### Diode Straps Connections

**STRAP 0:** Controls the on and off of radio. In applications where a toggle type ON/OFF switch is used, momentary OFF key can be omitted; connecting the strap will turn on the radio and vice versa. Must be connected to use momentary OFF key.

**STRAP 1, 2:** Selects the AM IF options.

**STRAP 2:** 12/24-hour clock select.

**STRAP 4:** 3/5kHz AM step size select.

**STRAP 5, 6:** FM IF offsets select.

	STRAP 0	STRAP 3	STRAP 4
Connected	Radio ON	12 hour	5kHz step
Open	Radio OFF	24 hour	3kHz step

### AM/FM IF Options:

	AM	STRAP 1	STRAP 2
455kHz		X	X
460kHz		X	✓
450kHz		✓	X
260kHz		✓	✓

	FM	STRAP 5	STRAP 6
10.7MHz		X	X
10.75MHz		X	✓
10.65MHz		✓	X
10.8MHz		✓	✓

X = No connection.  
✓ = Diode inserted.

### Indirect Features and Options

As indicated in Figure 10, there are a few options and indirect features provided via the help of a slave device, namely the Phase Lock Loop, DS8906N.

### Display Options

As mentioned above, the COP420L-HSB is MICRO-WIRE® compatible. Internal circuitry enables it to directly interface with all of National's serial input MICRO-WIRE compatible display drivers whether they are of a direct drive or multiplex drive format. On Figure 10 is a list of drivers available for the system. EN1 and EN2 are optional enable outputs meant for a dual display system in which EN3 will not be used. By dual display, it means that one display will be constantly showing time informa-

tion and the other showing frequency information. Whereas in conventional single display systems, the display shows both time and frequency information in a time-sharing method. The National system provides a time-prioritized display-sharing method. That is, whenever a tuning function is completed, the frequency information will stay on the display for eight seconds then time display will take over. This is achieved by using EN3 for the driver's enable logic.

**Control Outputs**

Six open collector outputs controlled by the COP420L-HSB are provided from DS8906N, the phase lock loop for controlling radio switching circuits.

**Radio ON/OFF:** A high from this output indicates that the radio should be switched on and vice versa.

**AM/FM:** Output for controlling the AM/FM bandswitch. A high level output indicates FM and a low indicates the AM band.

**MUTE:** For muting the audio output when performing any frequency related function. The output will go high prior to the frequency change except when doing fine tuning.

**ALARM ENABLE:** Active high output for turning on the alarm circuit at alarm time.

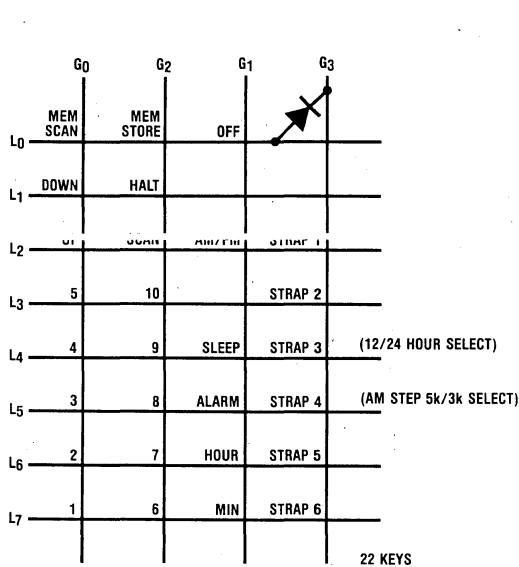
**50kHz IND:** For driving the 50kHz indicator in FM band or the LSB in a 5-digit display. Output is active high.

**MEM STORE IND:** For driving the memory store mode indicator. Output is active high.

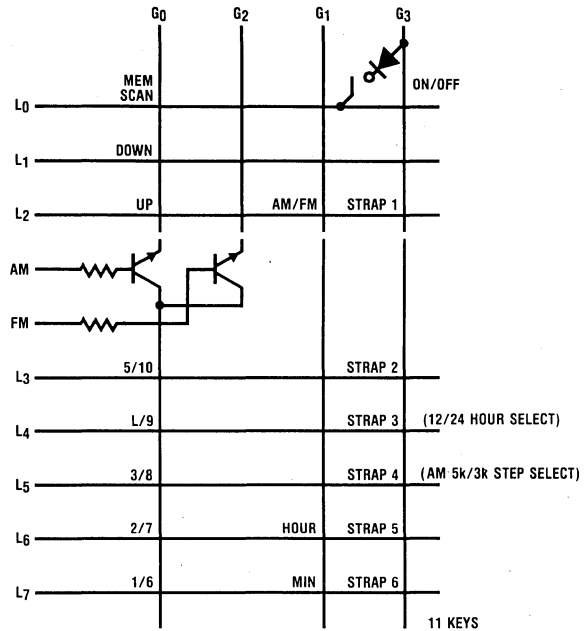
**Typical Implementation Alternatives**

A full keyboard or any portion of it can be implemented with various applications for features/functions vs. cost/size.

Figure 11 shows two keyboard configurations with 22-key and 11-key keyboards for a desk top/tuner system or auto-radio system respectively.



Desk Top DTR Keyboard



Car DTR Keyboard

Figure 11.





# COP440/COP441/COP442 and COP340/COP341/COP342 Single-Chip N-Channel Microcontrollers

## General Description

The COP440, COP441, COP442, COP340, COP341, and COP342 Single-Chip N-Channel Microcontrollers are members of the COPST<sup>™</sup> family, fabricated using N-channel, silicon gate MOS technology. These are complete microcontrollers with all system timing, internal logic, ROM, RAM, and I/O necessary to implement dedicated control functions in a variety of applications. Features include single supply operation, various output configuration options, and an instruction set, internal architecture, and I/O scheme designed to facilitate keyboard input, display output, and data manipulation. The COP440 is a 40-pin chip and the COP441 is a 28-pin version of the same circuit (12 I/O lines removed). The COP442 is a 24-pin version (4 more input lines removed). The COP340, COP341, COP342 are functional equivalents of the above devices respectively, but operate with an extended temperature range (-40°C to +85°C). Standard test procedures and reliable high-density fabrication techniques provide the medium to large volume customers with a customized controller oriented processor at a low end-product cost.

## Features

- Enhanced, more powerful instruction set
- 2k x 8 ROM, 160 x 4 RAM
- 35 I/O lines (COP440)
- Zero-crossing detect circuitry with hysteresis
- True multi-vectored interrupt from 4 selectable sources (plus restart)
- Four-level subroutine stack (in RAM)
- 4 μs cycle time
- Single supply operation (4.5V-6.3V)
- Programmable time-base counter
- Internal binary counter/register with MICROWIRE<sup>™</sup> compatible serial I/O
- General purpose and TRI-STATE<sup>®</sup> outputs
- TTL/CMOS compatible in and out
- LED drive capability
- MICROBUS<sup>™</sup> compatible
- Software/hardware compatible with other members of the COP400 family
- Extended temperature range devices COP340, COP341, COP342 (-40°C to +85°C)
- Compatible dual CPU device available (COP2440 series)

COPS, MICROBUS, and MICROWIRE are trademarks of National Semiconductor Corp.  
TRI-STATE is a registered trademark of National Semiconductor Corp.

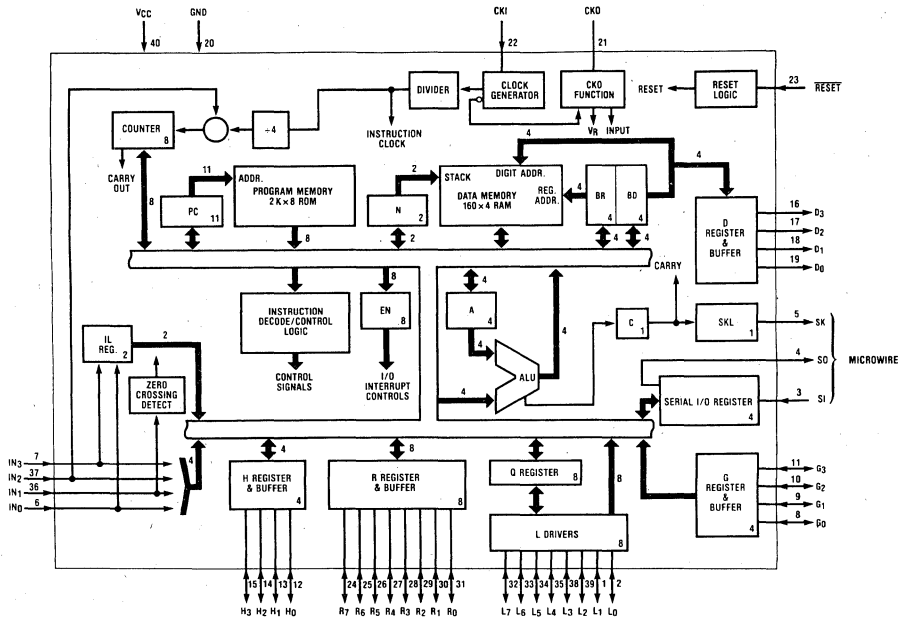


Figure 1. COP440 Block Diagram

## COP440/COP441/COP442

### Absolute Maximum Ratings

Voltage at Zero-Crossing Detect Pin Relative to GND	-1.2V to +15V
Voltage at Any Other Pin Relative to GND	-0.5V to +7V
Ambient Operating Temperature	0°C to +70°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C
Power Dissipation	0.75 Watt at 25°C 0.4 Watt at 70°C
Total Source Current	150 mA
Total Sink Current	75 mA

*Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

### DC Electrical Characteristics $0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ , $4.5\text{V} \leq V_{CC} \leq 6.3\text{V}$ unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Operating Voltage ( $V_{CC}$ )	Note 3	4.5	6.3	V
Power Supply Ripple	(peak to peak)		0.4	V
Operating Supply Current	(All inputs and outputs open) $T_A = 0^{\circ}\text{C}$ $T_A = 25^{\circ}\text{C}$ $T_A = 70^{\circ}\text{C}$		41 35 27	mA mA mA
Input Voltage Levels				
CKI Input Levels				
Crystal Input ( $\pm 16, \pm 8$ )				
Logic High ( $V_{IH}$ )	$V_{CC} = \text{Max.}$	2.5		V
Logic High ( $V_{IH}$ )	$V_{CC} = 5\text{V} \pm 5\%$	2.0		V
Logic Low ( $V_{IL}$ )		-0.3	0.4	V
Schmitt Trigger Input ( $\pm 4$ )				
Logic High ( $V_{IH}$ )		$0.7V_{CC}$		V
Logic Low ( $V_{IL}$ )		-0.3	0.6	V
RESET Input Levels	(Schmitt Trigger Input)			
Logic High		$0.7V_{CC}$		V
Logic Low		-0.3	0.6	V
Zero-Crossing Detect Input	See Figure 7			
Trip Point		-0.15	0.15	V
Logic High ( $V_{IH}$ ) Limit			12	V
Logic Low ( $V_{IL}$ ) Limit		-0.8		V
SO Input Level (Test Mode)		2.0	2.5	V
All Other Inputs				
Logic High	$V_{CC} = \text{Max.}$	2.5		V
Logic High	$V_{CC} = 5\text{V} \pm 5\%$	2.0		V
Logic Low		-0.3	0.8	V
Input Levels High Trip Option				
Logic High		3.6		V
Logic Low		-0.3	1.2	V
Input Capacitance			7.0	pF
Hi-Z Input Leakage		-1.0	+1.0	$\mu\text{A}$

**Note 1:** Duty Cycle =  $t_{WH}/(t_{WH} + t_{WO})$ .

**Note 2:** See Figure for additional I/O Characteristics.

**Note 3:**  $V_{CC}$  voltage change must be less than 0.5V in a 1ms period to maintain proper operation.

**Note 4:** Exercise great care not to exceed maximum device power dissipation limits when direct-driving LEDs (or sourcing similar loads) at high temperature.

## COP440/COP441/COP442

### DC Electrical Characteristics (Cont'd)

Parameter	Conditions	Min.	Max.	Units
Output Voltage Levels				
Standard Output				
TTL Operation				
Logic High ( $V_{OH}$ )	$I_{OH} = -100\mu A$	2.4		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 1.6mA$		0.4	V
CMOS Operation				
Logic High ( $V_{OH}$ )	$I_{OH} = -10\mu A$	$V_{CC} - 0.4$		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 10\mu A$		0.2	V
Output Current Levels				
Standard Output Source Current	$V_{CC} = 4.5V, V_{OH} = 2.4V$	-100	-650	$\mu A$
LED Direct Drive Output	$V_{CC} = 6V$			
Logic High ( $I_{OH}$ )	$V_{OH} = 2V$	-2.5	-17	mA
TRI-STATE® Output Leakage Current		-2.5	+2.5	$\mu A$
CKO Output				
Oscillator Output Option				
Logic High	$V_{OH} = 2V$	-0.2		mA
Logic Low	$V_{OL} = 0.4V$	0.4		mA
$V_R$ RAM Power Supply Option				
Supply current	$V_R = 3.3V$		3.0	mA
CKI Sink Current (RC Option)	$V_{IH} = 3.5V, V_{CC} = 4.5V$	2.0		mA
Input Current Levels				
Zero-Crossing Detect Input				
Resistance	$V_{IH} = 1.0V$	1.5	4.6	$k\Omega$
Input Load Source Current	$V_{IH} = 2.0V, V_{CC} = 4.5V$	14	230	$\mu A$
Total Sink Current Allowed				
All I/O Combined			75	mA
Each L, R Port			20	mA
Each D, G, H Port			10	mA
SO, SK			2.5	mA
Total Source Current Allowed				
All I/O Combined			150	mA
L Port			120	mA
L <sub>7</sub> -L <sub>4</sub>			70	mA
L <sub>3</sub> -L <sub>0</sub>			70	mA
Each L Pin			23	mA
All Other Output Pins			1.6	mA

## COP340/COP341/COP342

### Absolute Maximum Ratings

Voltage at Zero-Crossing Detect Pin Relative to GND	-1.2V to +15V
Voltage at Any Other Pin Relative to GND	-0.5V to +7V
Ambient Operating Temperature	-40°C to +85°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C
Power Dissipation	0.75 Watt at 25°C 0.25 Watt at 85°C
Total Source Current	150 mA
Total Sink Current	75 mA

*Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

### DC Electrical Characteristics

$-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 5.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Operating Voltage ( $V_{CC}$ )	Note 3	4.5	5.5	V
Power Supply Ripple	(peak to peak)		0.4	V
Operating Supply Current	(All inputs and outputs open)			
	$T_A = -40^{\circ}\text{C}$		54	mA
	$T_A = 25^{\circ}\text{C}$		35	mA
	$T_A = 85^{\circ}\text{C}$		25	mA
Input Voltage Levels				
CKI Input Levels				
Crystal Input ( $\pm 16, \pm 8$ )				
Logic High ( $V_{IH}$ )		2.2		V
Logic Low ( $V_{IL}$ )		-0.3	0.3	V
Schmitt Trigger Input ( $\pm 4$ )				
Logic High ( $V_{IH}$ )		0.7 $V_{CC}$		V
Logic Low ( $V_{IL}$ )		-0.3	0.4	V
RESET Input Levels	(Schmitt Trigger Input)			
Logic High		0.7 $V_{CC}$		V
Logic Low		-0.3	0.4	V
Zero-Crossing Detect Input	See Figure 7			
Trip Point		-0.15	0.15	V
Logic High ( $V_{IH}$ ) Limit			12	V
Logic Low ( $V_{IL}$ ) Limit		-0.8		V
SO Input Level (Test Mode)		2.2	2.4	V
All Other Inputs				
Logic High		2.2		V
Logic Low		-0.3	0.6	V
Input Levels High Trip Option				
Logic High		3.6		V
Logic Low		-0.3	1.2	V
Input Capacitance			7.0	pF
Hi-Z Input Leakage		-2.0	+2.0	$\mu\text{A}$

### COP340/COP341/COP342

#### DC Electrical Characteristics (Cont'd)

Parameter	Conditions	Min.	Max.	Units
Output Voltage Levels				
Standard Output				
TTL Operation				
Logic High ( $V_{OH}$ )	$I_{OH} = -100 \mu A$	2.4		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 1.6 mA$		0.4	V
CMOS Operation				
Logic High ( $V_{OH}$ )	$I_{OH} = -10 \mu A$	$V_{CC} - 0.5$		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 10 \mu A$		0.2	V
Output Current Levels				
Standard Output Source Current	$V_{CC} = 4.5V, V_{OH} = 2.4V$	-100	-800	$\mu A$
LED Direct Drive Output	$V_{CC} = 5V$ (Note 4)			
Logic High ( $I_{OH}$ )	$V_{OH} = 2V$	-1.5	-15	mA
TRI-STATE® Output Leakage Current		-5.0	+5.0	$\mu A$
CKO Output				
Oscillator Output Option				
Logic High	$V_{OH} = 2V$	-0.2		mA
Logic Low	$V_{OL} = 0.4V$	0.4		mA
$V_R$ RAM Power Supply Option				
Supply current	$V_R = 3.3V$		4.0	mA
CKI Sink Current (RC Option)	$V_{CC} = 4.5V, V_{IH} = 3.5V$	2.0		mA
Input Current Levels				
Zero-Crossing Detect Input				
Resistance	$V_{IH} = 1.0V$	1.4	4.6	k $\Omega$
Input Load Source Current	$V_{IH} = 2.0V, V_{CC} = 4.5V$	14	280	$\mu A$
Total Sink Current Allowed				
All I/O Combined			75	mA
Each L, R Port			20	mA
Each D, G, H Port			10	mA
SO, SK			2.5	mA
Total Source Current Allowed				
All I/O Combined			150	mA
L Port			120	mA
L <sub>7</sub> -L <sub>4</sub>			70	mA
L <sub>3</sub> -L <sub>0</sub>			70	mA
Each L Pin			23	mA
All Other Output Pins			1.6	mA

### AC Electrical Characteristics

COP440/COP441/COP442:  $0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 6.3\text{V}$  unless otherwise noted.

COP340/COP341/COP342:  $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 5.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Instruction Cycle Time — $t_E$		4.0	10	$\mu\text{s}$
CKI Frequency	$\div 16$ mode	1.6	4.0	MHz
	$\div 8$ mode	0.8	2.0	MHz
	$\div 4$ mode	0.4	1.0	MHz
Duty Cycle (Note 1)	$f_I = 4\text{MHz}$	30	60	%
Rise Time	$f_I = 4\text{MHz}$ external clock		60	ns
Fall Time	$f_I = 4\text{MHz}$ external clock		40	ns
CKI Using RC (Figure 9c)	$\div 4$ mode			
Frequency	$R = 15\text{k}\Omega \pm 5\%$ , $C = 100\text{pF} \pm 10\%$	0.5	1.0	MHz
Instruction Execution Time — $t_E$		4.0	8.0	$\mu\text{s}$
<b>INPUTS: (Figure 4)</b>				
SI				
$t_{\text{SETUP}}$		0.3		$\mu\text{s}$
$t_{\text{HOLD}}$		300		ns
All Other Inputs				
$t_{\text{SETUP}}$		1.7		$\mu\text{s}$
$t_{\text{HOLD}}$		300		ns
<b>OUTPUT PROPAGATION DELAY</b>				
	Test Condition: $C_L = 50\text{pF}$ , $V_{\text{OUT}} = 1.5\text{V}$			
CKO				
$t_{\text{pd}1}$ , $t_{\text{pd}0}$	Crystal Input		0.17	$\mu\text{s}$
$t_{\text{pd}1}$ , $t_{\text{pd}0}$	Schmitt Trigger Input		0.3	$\mu\text{s}$
SO, SK				
$t_{\text{pd}1}$ , $t_{\text{pd}0}$	$R_L = 2.4\text{k}\Omega$		1.0	$\mu\text{s}$
All Other Outputs	$R_L = 5.0\text{k}\Omega$		1.4	$\mu\text{s}$
<b>MICROBUS™ TIMING</b>				
	$C_L = 100\text{pF}$ , $V_{CC} = 5\text{V} \pm 5\%$ TRI-STATE® outputs			
<b>Read Operation (Figure 2a)</b>				
Chip Select Stable Before $\overline{\text{RD}}$ — $t_{\text{CSR}}$		65		ns
Chip Select Hold Time for $\overline{\text{RD}}$ — $t_{\text{RCS}}$		20		ns
$\overline{\text{RD}}$ Pulse Width — $t_{\text{RR}}$		400		ns
Data Delay from $\overline{\text{RD}}$ — $t_{\text{RD}}$			375	ns
$\overline{\text{RD}}$ to Data Floating — $t_{\text{DF}}$			250	ns
<b>Write Operation (Figure 2b)</b>				
Chip Select Stable Before $\overline{\text{WR}}$ — $t_{\text{CSW}}$		65		ns
Chip Select Hold Time for $\overline{\text{WR}}$ — $t_{\text{WCS}}$		20		ns
$\overline{\text{WR}}$ Pulse Width — $t_{\text{WW}}$		400		ns
Data Set-Up Time for $\overline{\text{WR}}$ — $t_{\text{DW}}$		320		ns
Data Hold Time for $\overline{\text{WR}}$ — $t_{\text{WD}}$		100		ns
INTR Transition Time from $\overline{\text{WR}}$ — $t_{\text{WI}}$			700	ns

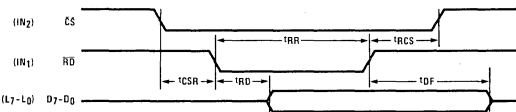


Figure 2a. MICROBUS™ Read Operation Timing

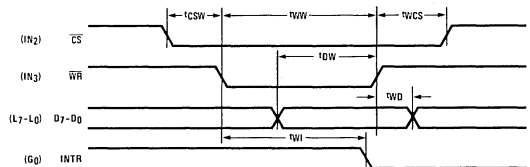
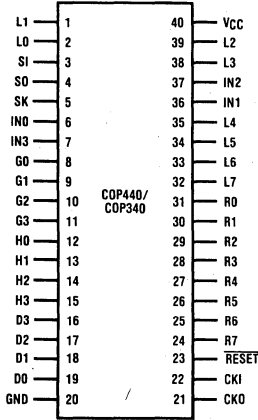
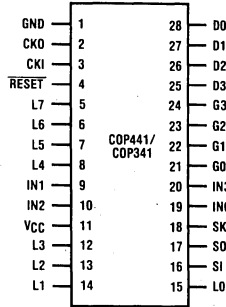


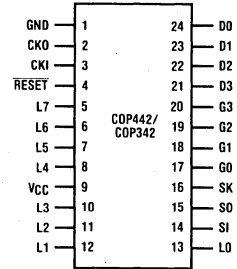
Figure 2b. MICROBUS Write Operation Timing



Order Number COP440N, COP340N  
NS Package N40A



Order Number COP441N, COP341N  
NS Package N28A



Order Number COP442N, COP342N  
NS Package N24A

Figure 3. Connection Diagrams

Pin	Description	Pin	Description
L <sub>7</sub> -L <sub>0</sub>	8-bit bidirectional I/O port with TRI-STATE®	CKI	System oscillator input
G <sub>3</sub> -G <sub>0</sub>	4-bit bidirectional I/O port	CKO	System oscillator output (or general purpose input or RAM power supply)
D <sub>3</sub> -D <sub>0</sub>	4-bit general purpose output port	RESET	System reset input
IN <sub>3</sub> -IN <sub>0</sub>	4-bit general purpose input port (not available on COP442/COP342)	V <sub>CC</sub>	Power supply
SI	Serial input	GND	Ground
SO	Serial output (or general purpose output)	H <sub>3</sub> -H <sub>0</sub>	4-bit bidirectional I/O port (COP440/COP340 only)
SK	Logic-controlled clock (or general purpose output)	R <sub>7</sub> -R <sub>0</sub>	8-bit bidirectional I/O port with TRI-STATE® (COP440/COP340 only)

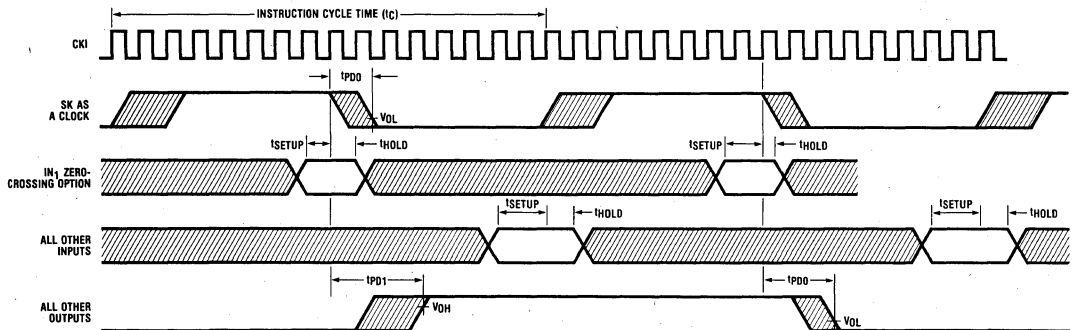


Figure 4. Input/Output Timing Diagrams (Divide by 16 Mode)

## Functional Description

The block diagram of the COP440 is shown in Figure 1. Data paths are illustrated in simplified form to depict how the various logic elements communicate with each other in implementing the instruction set of the device. Positive logic is used. When a bit is set, it is a logic "1" (greater than 2.0 volts). When a bit is reset, it is a logic "0" (less than 0.8 volts).

### Program Memory

Program Memory consists of a 2,048 byte ROM. As can be seen by an examination of the COP440 instruction set, these words may be program instructions, constants, or ROM addressing data. Because of the special characteristics associated with the JP, JSRP, JID, LQID, and LID instructions, ROM must often be thought of as being organized into 32 pages of 64 words each.

ROM addressing is accomplished by an 11-bit PC register. Its binary value selects one of the 2,048 8-bit words contained in ROM. A new address is loaded into the PC register during each instruction cycle. Unless the instruction is a transfer of control instruction, the PC register is loaded with the next sequential 11-bit binary count value.

ROM instruction words are fetched, decoded and executed by the Instruction Decode, Control and Skip Logic circuitry.

### Data Memory

Data memory consists of a 640-bit RAM, organized as 10 data registers of 16 4-bit digits. RAM addressing is implemented by an 8-bit B register whose upper 4 bits (Br) select 1 of 10 (0-9) data registers and lower 4 bits (Bd) select 1 of 16 4-bit digits in the selected data register. While the 4-bit contents of the selected RAM digit (M) is usually loaded into, or from, or exchanged with the A register (accumulator), it may also be loaded into or from the Q latches, L port, R port, EN register, and T counter (internal time base counter). RAM may also be loaded from 4 bits of a ROM word. RAM addressing may also be performed directly to the lower 8 registers by the LDD and XAD instructions based upon the 7-bit contents of the operand field of these instructions. The Bd register also serves as a source register for 4-bit data sent directly to the D outputs. RAM register 8 (Br = 8) also serves as a subroutine stack. Note that it is possible, but not recommended, to alter the contents of the stack by normal data memory access commands.

### Internal Logic

The 4-bit A register (accumulator) is the source and destination register for most I/O arithmetic, logic, and data memory access operations. It can also be used to load the Br and Bd portions of the B register, N register, to load and input 4 bits of the 8-bit Q latch, EN register, or T counter, to input 4 bits of a ROM word, L or R I/O port data, to input 4-bit G, H, or IN ports, and to perform data exchanges with the SIO register. The accumulator is cleared upon reset.

A 4-bit adder performs the arithmetic and logic functions of the COP440, storing its results in A. It also outputs a carry bit to the 1-bit C register, most often employed to indicate arithmetic overflow. The C register, in conjunction with the XAS instruction and the EN register, also serves to control the SK output. C can be outputted directly to SK or can enable SK to be a sync clock each instruction cycle time. (See XAS instruction and EN register description, below.)

The 8-bit T counter is a binary up counter which can be loaded to and from M and A. The input to this counter is software selectable from two sources: the first coming from a divide-by-four prescaler (from instruction cycle frequency) thus providing a 10-bit time base counter; the second coming from IN<sub>2</sub> input, changing the T counter into an 8-bit external event counter (see EN register below). In this mode, a low-going pulse ("1" to "0") of at least 2 instruction cycles wide will increment the counter. When the counter overflows, an overflow flag will be set (see SKT instruction below) and an interrupt signal will be sent to processor X. The T counter is cleared on reset.

Four general-purpose inputs, IN<sub>3</sub>-IN<sub>0</sub>, are provided; IN<sub>1</sub>, IN<sub>2</sub> and IN<sub>3</sub> may be selected, by a mask-programmable option, as Read Strobe, Chip Select, and Write Strobe inputs, respectively, for use in MICROBUS™ applications; IN<sub>1</sub>, by another mask-programmable option, can be selected as a true zero-crossing detector with the output triggering an interrupt or being interrogated by an instruction. These two mask-programmable options are mutually exclusive.

The D register provides 4 general-purpose outputs and is used as the destination register for the 4-bit contents of Bd.

The G register contents are outputs to a 4-bit general-purpose bidirectional I/O port. G<sub>0</sub> may be mask-programmed as an output for MICROBUS applications.

The H register contents are outputs to a 4-bit general-purpose bidirectional I/O port.

The Q register is an internal, latched, 8-bit register, used to hold data loaded to or from M and A, as well as 8-bit data from ROM. Its contents are outputted to the L I/O ports when the L drivers are enabled under program control. With the MICROBUS option selected, Q can also be loaded with the 8-bit contents of the L I/O ports upon the occurrence of a write strobe from the host CPU. Note that unlike most other COPST™ controllers, Q is cleared on reset.

The 8 L drivers, when enabled, output the contents of latched Q data to the L I/O ports. Also, the contents of L may be read directly into A and M. As explained above, the MICROBUS option allows L I/O port data to be latched into the Q register. The L I/O port can be directly connected to the segments of a multiplexed LED display (using the LED Direct Drive output configuration option) with Q data being outputted to the Sa-Sg and decimal point segments of the display.

The R register, when enabled, outputs to an 8-bit general-purpose, bidirectional, I/O port.



The SIO register functions as a 4-bit serial-in/serial-out shift register for MICROWIRE™ I/O and COPS™ peripherals, or as a binary counter (depending on the contents of the EN register; see EN register description, below). Its contents can be exchanged with A, allowing it to input or output a continuous serial data stream.

The XAS instruction copies the C flag into the SKL latch. In the counter mode, SK is the output of SKL; in the shift register mode, SK outputs SKL ANDed with the instruction cycle clock.

The 2-bit N register is a stack pointer to the data memory register 8 where the subroutine return address is located. It points to the next location where the address may be stored and increments by 1 after each push of the stack, and decrements by 1 before each pop. The N register can be accessed by exchanging its value with A and is cleared on reset. The stack is 4 addresses deep, 12 bits wide, and does not check for overflow or empty conditions. The RAM digit locations where the addresses are stored are shown in Figure 5. The LSBs of the addresses are at digits 0, 4, 8, and 12. The MSBs of digits 2, 6, 10, and 14 contain an interrupt status bit (see Interrupt description, below). The four unused digits (3, 7, 11, and 15) can be used as general data storage. When a subroutine call or interrupt occurs, an 11-bit return address and an interrupt status bit are stored in the stack. The N register is then incremented. When a RET or RETSK instruction is executed, the N register is decremented and then the return address is fetched and loaded into the program counter. The address and interrupt status bits remain in the stack, but will be overwritten when the next subroutine call or interrupt occurs.

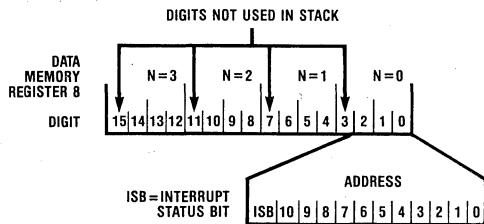


Figure 5. Subroutine Return Address Stack Organization

The EN register is an internal 8-bit register loaded under program control by the LEI instruction (lower 4 bits) or by the CAME instruction. The state of each bit of this register selects or deselects the particular feature associated with each bit of the EN register:

0. The least significant bit of the enable register, EN<sub>0</sub>, selects the SIO register as either a 4-bit shift register or a 4-bit binary counter. With EN<sub>0</sub> set, SIO is an asynchronous binary counter, decrementing its value by one upon each low-going pulse ("1" to "0") occurring on the SI input. Each pulse must be at least two instruction cycles wide. SK outputs the value of SKL. The SO output is equal to the value of EN<sub>3</sub>. With EN<sub>0</sub> reset, SIO is a serial shift register left shifting 1 bit each instruction cycle time. The data present at SI goes into the least significant bit of SIO. SO can be enabled to output the most significant bit of SIO each cycle time. The SK output becomes a logic-controlled clock.
1. With EN<sub>1</sub> set, interrupt is enabled with EN<sub>4</sub> and EN<sub>5</sub> selecting the interrupt source. Immediately following an interrupt, EN<sub>1</sub> is reset to disable further interrupts.
2. With EN<sub>2</sub> set, the L drivers are enabled to output the data in Q to the L I/O port. Resetting EN<sub>2</sub> disables the L drivers, placing the L I/O port in a high-impedance input state. A special feature of the COP440 and COP441 is that the MICROBUS™ option will change the function of this bit to disable any writing into G<sub>0</sub> when EN<sub>2</sub> is set.
3. EN<sub>3</sub>, in conjunction with EN<sub>0</sub>, affects the SO output. With EN<sub>0</sub> set (binary counter option selected) SO will output the value loaded into EN<sub>3</sub>. With EN<sub>0</sub> reset (serial shift register option selected), setting EN<sub>3</sub> enables SO as the output of the SIO shift register, outputting serial shifted data each instruction time. Resetting EN<sub>3</sub> with the serial shift register option selected disables SO as the shift register output; data continues to be shifted through SIO and can be exchanged with A via an XAS instruction but SO remains set to "0." Table 1 below provides a summary of the modes associated with EN<sub>3</sub> and EN<sub>0</sub>.

Table 1. Enable Register Modes — Bits EN<sub>3</sub> and EN<sub>0</sub>

EN <sub>3</sub>	EN <sub>0</sub>	SIO	SI	SO	SK
0	0	Shift Register	Input to Shift Register	0	If SKL = 1, SK = Clock If SKL = 0, SK = 0
1	0	Shift Register	Input to Shift Register	Serial Out	If SKL = 1, SK = Clock If SKL = 0, SK = 0
0	1	Binary Counter	Input to Binary Counter	0	If SKL = 1, SK = 1 If SKL = 0, SK = 0
1	1	Binary Counter	Input to Binary Counter	1	If SKL = 1, SK = 1 If SKL = 0, SK = 0

- 4. EN<sub>5</sub> and EN<sub>4</sub> select the source of the interrupt signal.
- 5. The possible sources are as follows:

EN <sub>5</sub>	EN <sub>4</sub>	Interrupt Source
0	0	IN <sub>1</sub> (low-going pulse)
0	1	CKO input (if mask-programmed as an input)
1	0	Zero-crossing (or IN <sub>1</sub> level transition)
1	1	T counter overflows

EN<sub>4</sub> determines the interrupt routine location.

- 6. With EN<sub>6</sub> set, the internal 8-bit T counter will use IN<sub>2</sub> as its input. With EN<sub>6</sub> reset, the input to the T counter is the output of a divide by four prescaler (from instruction cycle frequency), thus providing a 10-bit time-base counter.
- 7. With EN<sub>7</sub> set, the R outputs are enabled; if EN<sub>7</sub> = 0, the R outputs are disabled.

**Interrupt**

The following features are associated with the interrupt procedure and protocol and must be considered by the programmer when utilizing interrupts.

- a. The interrupt, once acknowledged as explained below, pushes the next sequential program counter address (PC + 1) together with an interrupt status bit, onto the program counter stack residing in data memory. Any previous contents at the bottom of the stack are lost. The program counter is set to hex address 0FF (the last word of page 3) and EN<sub>1</sub> is reset. If EN<sub>4</sub> is reset, the next program address is hex 100; if EN<sub>4</sub> is set, the next program address is hex 300; thus providing a different interrupt location for different interrupt sources.
- b. An interrupt will be acknowledged only after the following conditions are met:
  - 1. EN<sub>1</sub> has been set.
  - 2. For an external interrupt input, the signal pulse must be at least two instruction cycles wide.
  - 3. A currently executing instruction has been completed.
  - 4. All successive transfer of control instructions and successive LBIs have been completed (e.g., if the main program is executing a JP instruction which transfers program control to another JP instruction, the interrupt will not be acknowledged until the second JP instruction has been executed).
- c. The instruction at hex address 0FF must be a NOP.
- d. A CAME or LEI instruction may be put immediately before the RET instruction to re-enable interrupts.
- e. If the interrupt signal source is being changed, the interrupt must be disabled prior to, or at, the same time with the change to avoid false interrupts. An

interrupt may be enabled only if the interrupt source is not changing. A sample code for changing the interrupt source and enabling the interrupt is as follows:

```

CAME      ; disable interrupt & alter interrupt source
SMB 1     ; set interrupt enable bit
CAME      ; enable interrupt
    
```

- f. An interrupt status bit is stored together with the return address in the stack. The status bit is set if an interrupt occurs at a point in the program where the next instruction is to be skipped; upon returning from the interrupt routine, this set status bit will cause the next instruction to be skipped. Subroutine and interrupt nesting inside interrupt routines are allowed. Note that this differs from the COP420/420C/420L/444L series.

**MICROBUS™ Interface  
(not available in COP442, COP342)**

The COP440 series have an option which allows them to be used as peripheral microprocessor devices, inputting and outputting data from and to a host microprocessor (μP). IN<sub>1</sub>, IN<sub>2</sub> and IN<sub>3</sub> general purpose inputs become MICROBUS-compatible read-strobe, chip-select, and write-strobe lines, respectively. IN<sub>1</sub> becomes RD — a logic “0” on this input will cause Q latch data to be enabled to the L ports for input to the μP. IN<sub>2</sub> becomes CS — a logic “0” on this line selects the COPS™ processor as the μP peripheral device by enabling the operation of the RD and WR lines and allows for the selection of one of several peripheral components. IN<sub>3</sub> becomes WR — a logic “0” on this line will write bus data from the L ports to the Q latches for input to the COPS processor. G<sub>0</sub> becomes INTR, a “ready” output, reset by a write pulse from the μP on the WR line, providing the “handshaking” capability necessary for asynchronous data transfer between the host CPU and the COPS processor. G<sub>n</sub> output can be separated from other G outputs by the EN<sub>2</sub> bit (see EN description above).

This option has been designed for compatibility with National’s MICROBUS — a standard interconnect system for 8-bit parallel data transfer between MOS/LSI CPUs and interfacing devices. (See MICROBUS National Publication.) The functional and timing relationships between the COPS processor signal lines affected by this option are as specified for the MICROBUS interface, and are given in the AC electrical characteristics and shown in the timing diagrams (Figure 2). Connection of the COP440 to the MICROBUS is shown in Figure 6.

Note: TRI-STATE® outputs must be used on L port.

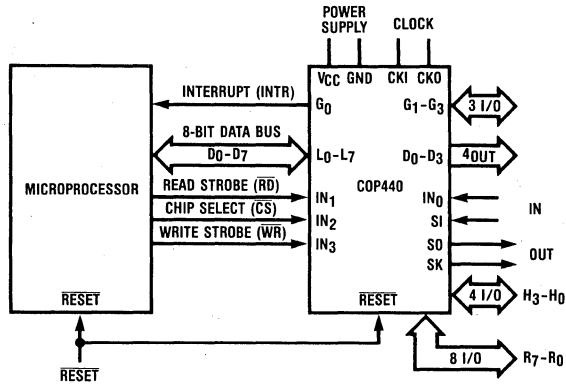


Figure 6. MICROBUS™ Option Interconnect

**Zero-Crossing Detection**  
(not available on the COP442, COP342)

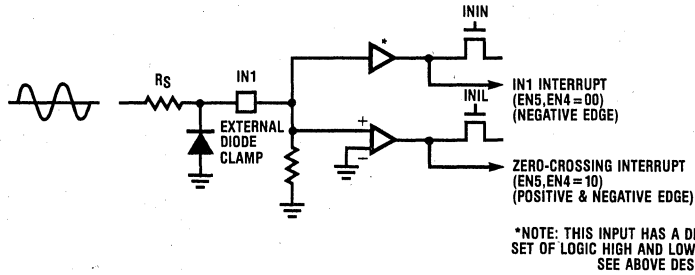
The following features are associated with the IN<sub>1</sub> pin: ININ and INIL instructions input the state of IN<sub>1</sub> to A<sub>1</sub>; IN<sub>1</sub> interrupt generates an interrupt pulse when a low-going transition ("1" to "0") occurs on IN<sub>1</sub>; zero-crossing interrupt generates an interrupt pulse when an IN<sub>1</sub> transition occurs (both "1" to "0" and "0" to "1").

If the zero-crossing detector is mask-programmed in (see Figure 7a), the INIL instruction and zero-crossing interrupt will input the state of IN<sub>1</sub> through the true zero-crossing detector ("1" if input > 0V, "0" if input < 0V). The ININ instruction and IN<sub>1</sub> interrupt will then have unique logic HIGH and LOW levels depending on the IN port input level chosen. If normal (TTL) level is chosen, logic HIGH level is 3.0V (3.3V for COP340/341) and logic LOW level is 0.8V (0.6V for COP340/341); if high trip level is chosen, logic HIGH level is 5.4V and logic LOW level is 1.2V. If the zero-crossing detector is not mask-programmed in

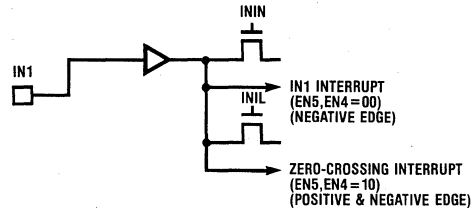
(see Figure 7b), IN<sub>1</sub> will have logic HIGH and LOW levels that are defined for the IN port (see option list).

The zero-crossing detector input contains a small hysteresis (50mV typical) to eliminate signal noise, and is not a high impedance input but contains a resistive load to ground. Since this input can withstand a voltage range of -0.8V to +12V, an external clamping diode is needed for most input signals, as shown in Figure 7a, to limit the voltage below ground. An external resistor, R<sub>S</sub> may be needed for the following two cases:

- Input signal exceeds 12V; R<sub>S</sub> and the internal resistor act as a voltage divider to reduce the voltage at the input pin to below 12V.
- Signal comes from a low impedance source; when the voltage at the pin is clamped to -0.7V by the forward bias voltage of an external diode, R<sub>S</sub> limits the current going through the diode.



a. Zero-Crossing Detect Logic Option



b. IN, without Zero-Crossing Detect Logic

Figure 7. IN<sub>1</sub> Mask-Programmable Options

### Initialization

The reset logic, internal to the COP440, will initialize the device upon power-up if the power supply rise time is less than 1ms and greater than 1 $\mu$ s. If the power supply rise time is greater than 1ms, the user must provide an external RC network and diode to the **RESET** pin as in Figure 8. The **RESET** pin is configured as a Schmitt trigger input. If not used, it should be connected to  $V_{CC}$ . Initialization will occur whenever a logic "0" is applied to the **RESET** input, provided it stays low for at least three instruction cycle times.

Upon initialization, the PC register is cleared to 0 (ROM address 0) and the A, B, C, D, EN, G, H, IL, L, N, Q, R, and T registers are cleared. The SK output is enabled as a SYNC output by setting the SKL latch, thus providing a clock. RAM (data memory and stack) is not cleared. The first instruction at address 0 must be a CLRA.

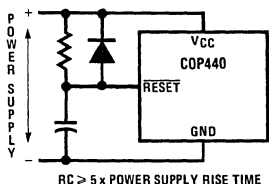


Figure 8. Power-Up Clear Circuit

### Oscillator

There are three basic clock oscillator configurations available, as shown by Figure 9.

**a. Crystal Controlled Oscillator.** CKI and CKO are connected to an external crystal. The cycle frequency equals the crystal frequency divided by 16 (optional by 8). Thus a 4 MHz crystal with the divide-by-16 option selected will give a 250 kHz cycle frequency (4 $\mu$ s instruction cycle time).

- b. External Oscillator.** CKI is an external clock input signal. The external frequency is divided by 16 (optional by 8 or 4) to give the cycle frequency. If the divide-by-4 option is selected, the CKI input level is the Schmitt-trigger level. CKO is now available to be used as the RAM power supply ( $V_R$ ) or as a general purpose input.
- c. RC Controlled Oscillator.** CKI is configured as a single pin RC controlled Schmitt trigger oscillator. The cycle frequency equals the oscillation frequency divided by 4. CKO is available for non-timing functions.

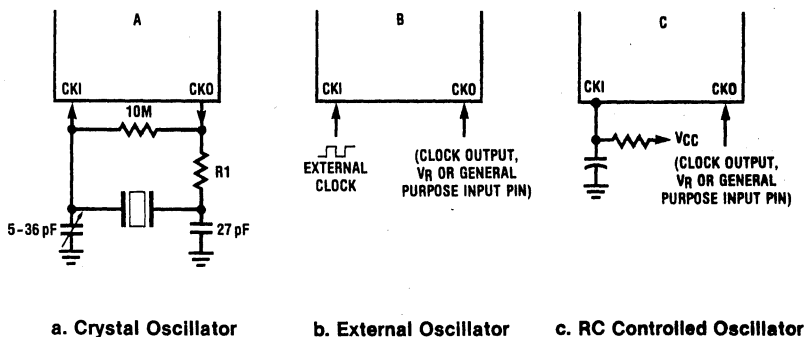
### CKO Pin Options

As an option, CKO can be an oscillator output. In a crystal controlled oscillator system, this signal is used as an output to the crystal network. As another option, CKO can be an interrupt input or a general purpose input, reading into bit 2 of A (accumulator) through the INIL instruction. As another option, CKO can be a RAM power supply pin ( $V_R$ ), allowing its connection to a standby/backup power supply to maintain the data integrity of RAM registers 0-3 with minimum power drain when the main supply is inoperative or shut down to conserve power. Using either of the two latter options is appropriate in applications where the system configuration does not require use of the CKO pin for timing functions.

### RAM Keep-Alive Option

Selecting CKO as the RAM power supply ( $V_R$ ) allows the user to shut off the chip power supply ( $V_{CC}$ ) and maintain data in the lower 4 registers of the RAM. To insure that RAM data integrity is maintained, the following conditions must be met:

1. **RESET** must go low before  $V_{CC}$  goes below spec during power-off;  $V_{CC}$  must be within spec before **RESET** goes high on power-up.
2. When  $V_{CC}$  is on,  $V_R$  must be within the operating voltage range of the chip, and within 1 volt of  $V_{CC}$ .
3.  $V_R$  must be  $\geq 3.3$ v with  $V_{CC}$  off.



a. Crystal Oscillator

b. External Oscillator

c. RC Controlled Oscillator

#### Crystal Oscillator

Crystal Value	R <sub>1</sub>
4 MHz	1k
3.58 MHz	1k
2.10 MHz	2k

#### RC Controlled Oscillator

R (k $\Omega$ )	C (pF)	Instruction Execution Time ( $\mu$ s)
13	100	5.0 $\pm$ 20%
6.8	220	5.3 $\pm$ 23%
8.2	300	8.0 $\pm$ 22%
22	100	8.2 $\pm$ 17%

Note: 5k $\Omega$  < R < 50k $\Omega$   
50 pF < C < 360 pF

Figure 9. COP440/441/442 Oscillators

## I/O Options

COP440 inputs have the following optional configurations, illustrated in Figure 10:

- An on-chip depletion load device to  $V_{CC}$ .
- A Hi-Z input which must be driven to a "1" or "0" by external components.
- A resistive load to GND for the zero-crossing input option ( $IN_1$  only).

COP440 outputs have the following optional configurations:

- Standard** — an enhancement-mode device to ground in conjunction with a depletion-mode device to  $V_{CC}$ , compatible with TTL and CMOS input requirements. Available on SO, SK, D, G, and H outputs.
  - Open-Drain** — an enhancement-mode device to ground only, allowing external pull-up as required by the user's application. Available on SO, SK, D, G, L, H, and R outputs.
  - Push-Pull** — An enhancement-mode device to ground in conjunction with a depletion-mode device paralleled by an enhancement-mode device to  $V_{CC}$ . This configuration has been provided to allow for fast rise and fall times when driving capacitive loads. Available on SO and SK outputs only.
  - Standard L,R** — same as d., but may be disabled. Available on L and R outputs only (disabled on reset).
  - LED Direct Drive** — an enhancement-mode device to ground and  $V_{CC}$  together with a depletion device to  $V_{CC}$  meeting the typical current sourcing requirements of the segments of an LED display. The sourcing devices are clamped to limit current flow. These devices may be turned off under program control (See Functional Description, EN Register), placing the output in a high-impedance state to provide required LED segment blanking for a multiplexed display. Available on L outputs only.
- Notes:**
- When the driver is disabled, the depletion device may cause the output to settle down to an intermediate level between  $V_{CC}$  and GND. This voltage cannot be relied upon as a "1" level when reading the L inputs. The external signal must drive it to a "1" level.
  - Much power is dissipated by this driver in driving an LED. Care must be taken to limit the power dissipation of the chip to within the absolute maximum ratings specified.
- TRI-STATE® Push-Pull** — an enhancement-mode device to ground and  $V_{CC}$ . These outputs are TRI-STATE outputs, allowing for connection of these outputs to a data bus shared by other bus drivers. Available on L and R outputs only (in TRI-STATE mode on reset).
  - Push-Pull R** — same as f., but may be disabled. Available on R outputs only.
  - Additional depletion pull-up** — a depletion load to  $V_{CC}$  with the same current sourcing capability as the input load a., in addition to the output drive chosen. Available on L and R outputs only. *This device cannot be disabled*; therefore, open-drain outputs with "1" output and TRI-STATE outputs do not show high-impedance characteristics. This device is useful in applications where a pull-up with low source current is desired, e.g., reading keyboards and switches.
- The above input and output configurations share common enhancement-mode and depletion-mode devices. Specifically, all configurations use one or more of six devices (numbered 1-6 respectively). Minimum and maximum current ( $I_{OUT}$  and  $V_{OUT}$ ) curves are given in Figures 11 and 12 for each of these devices to allow the designer to effectively use these I/O configurations in designing a COP440 system.

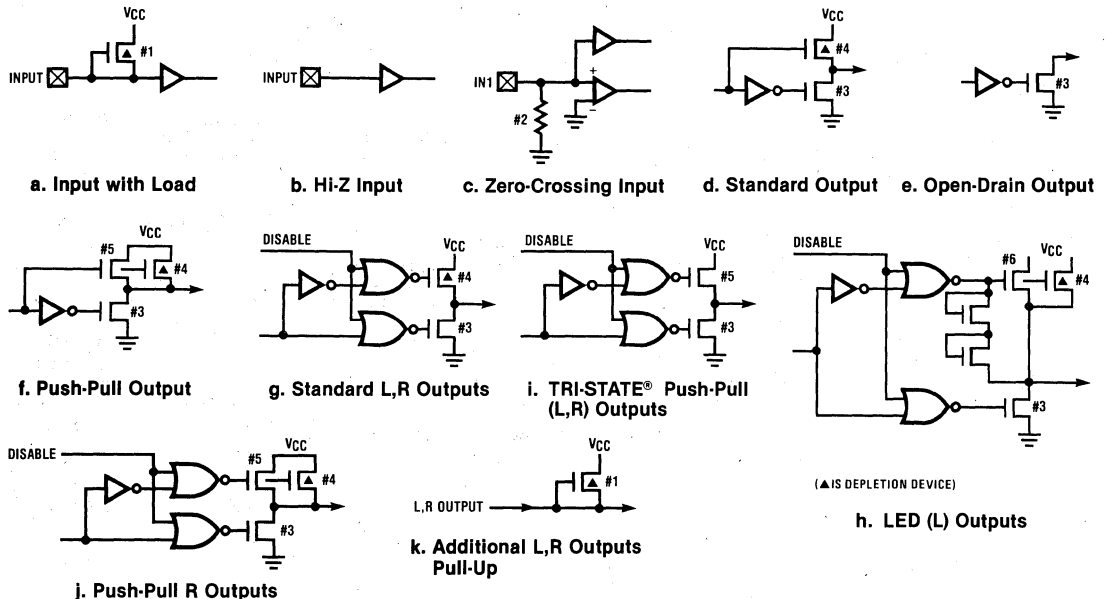
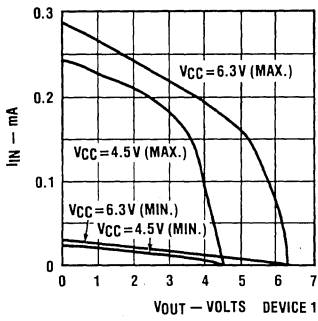
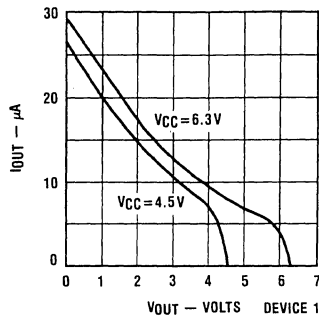


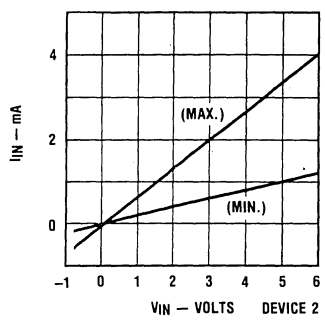
Figure 10. Input/Output Configurations



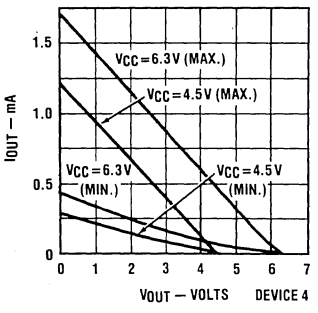
a. Input Load Source Current



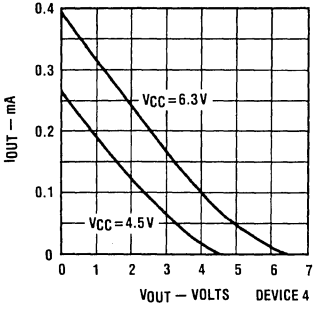
b. Input Load Minimum Source Current



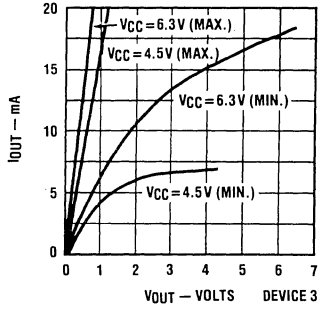
c. Zero-Crossing Detect Input Current



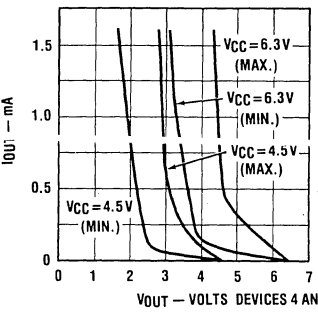
d. Standard Output Source Current



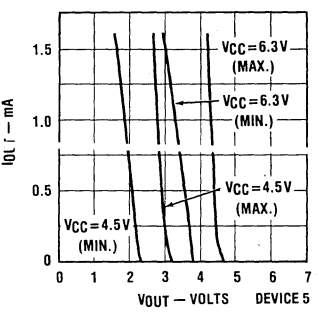
e. Standard Output Minimum Source Current



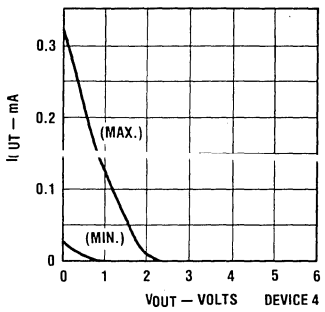
f. Output Sink Current



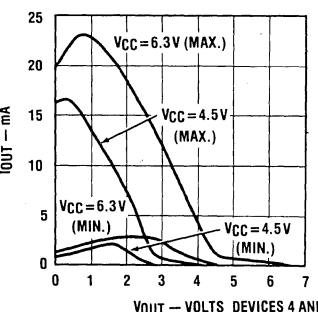
g. Push-Pull Source Current



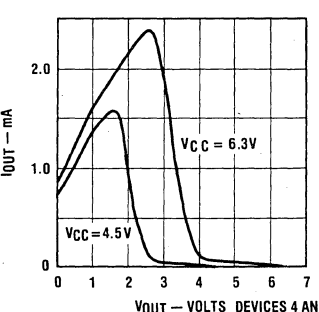
h. TRI-STATE Output Source Current



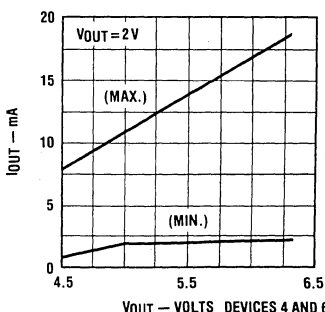
i. Depletion Load OFF Current



j. LED Output Source Current

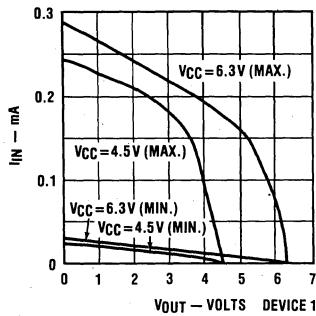


k. LED Output Minimum Source Current

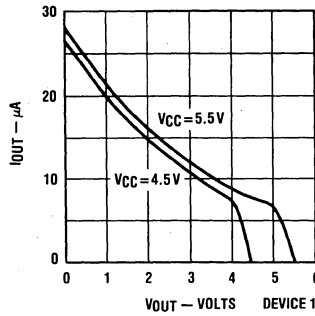


l. LED Output Direct LED Drive

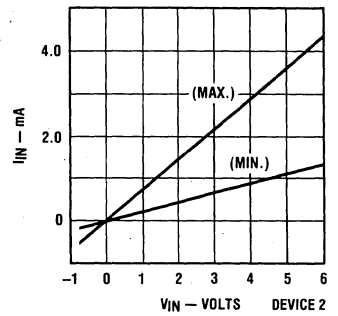
Figure 11. COP440/441/442 I/O Characteristics



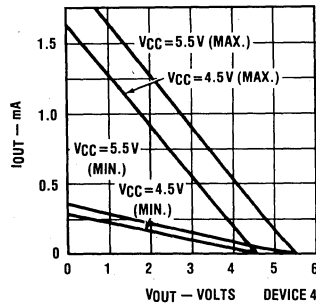
a. Input Load Source Current



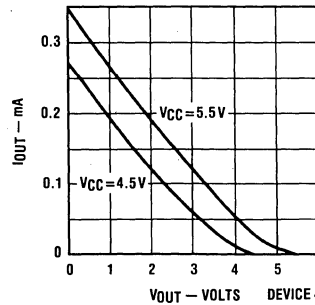
b. Input Load Minimum Source Current



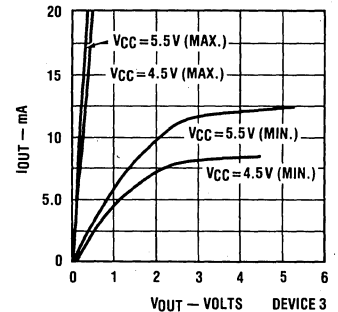
c. Zero-Crossing Detect Input Current



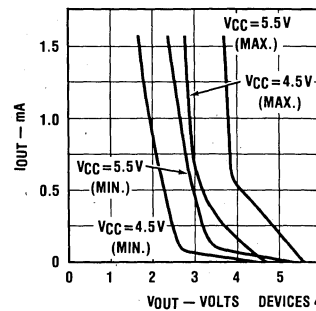
d. Standard Output Source Current



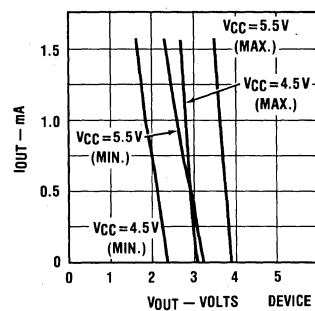
e. Standard Output Minimum Source Current



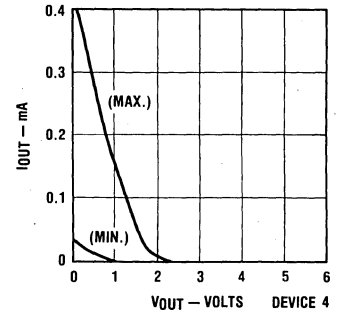
f. Output Sink Current



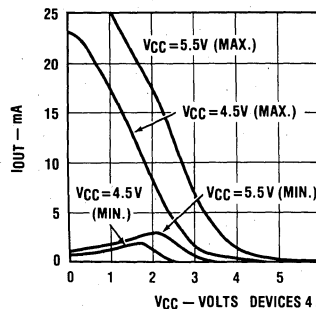
g. Push-Pull Source Current



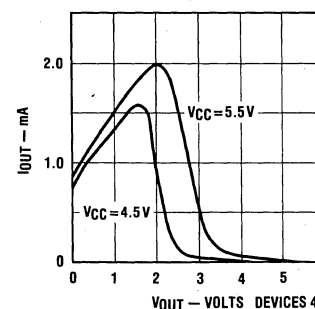
h. TRI-STATE Output Source Current



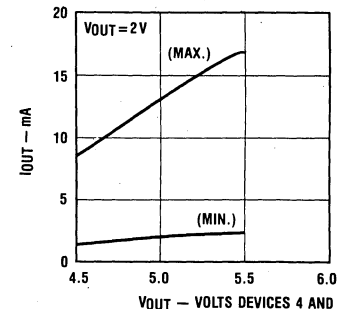
i. Depletion Load OFF Current



j. LED Output Source Current



k. LED Output Minimum Source Current



l. LED Output Direct LED Drive

Figure 12. COP340/341/342 I/O Characteristics

## Power Dissipation

In order not to damage the device by exceeding the absolute maximum power dissipation rating, the amount of power dissipated inside the chip must be carefully controlled. As an example, an application uses a COP440 in a room temperature (25°C) environment with a  $V_{CC}$  power supply of 6V; IN and SI inputs have internal loads; G and D ports drive loads that may sink up to 2mA into the chip; H port with standard output option reads switches; L port with the LED option drives a multiplexed seven-segment display; R, SO and SK drive MOS inputs that do not source or sink any current.

- a. At 25°C, maximum power dissipation allowed = 750mW.
- b. Power dissipation by chip except I/O =  $I_{CC} \times V_{CC} = 35\text{mA} \times 6\text{V} = 210\text{mW}$ .
- c. Maximum power dissipation by IN, SI =  $5 \times 0.3\text{mA} \times 6\text{V} = 9\text{mW}$
- d. G and D ports are sinking current from external loads; maximum output voltage with 2mA sink current is less than 0.4V. Power dissipation by G and D ports =  $2\text{mA} \times 0.4\text{V} \times 8 = 6.4\text{mW}$
- e. Maximum power dissipation by H port =  $4 \times 1.5\text{mA} \times 6\text{V} = 36\text{mW}$
- f. When the seven segments of the LED are turned on, the output voltage is about 2V, so that the segment current is 17mA. Power dissipation by L port =  $7 \times 17\text{mA} \times (6\text{V} - 2\text{V}) = 476\text{mW}$

This power dissipation caused by driving LEDs is usually the highest among the various sources.

- g. R, SO, and SK do not dissipate any significant amount of power because they do not need to source or sink any current.

Total power dissipation (TPD) inside the device is the sum of items b through g above.

$$\text{TPD} = 210 + 9 + 6 + 36 + 476\text{mW} = 737\text{mW}$$

This is within the 750mW limit at room temperature. If this application has to operate at 70°C, then the power dissipation must be reduced to meet the limit at that temperature. Some ways to achieve this would be to limit the LED current or to use an external LED driver.

At 70°C the absolute maximum power dissipation rating drops to 400mW. The user must be careful not to exceed this value.

## COP440 Series Devices

If the COP440 is bonded as a 28- or 24-pin device, it becomes the COP441 or COP442, respectively, as illustrated in Figure 3. Note that the COP441 and COP442 do not include H and R ports. In addition, the COP442 does not include IN inputs; use of this option precludes the use of the IN options, the interrupt feature with IN as input, the zero-crossing detect option, IN<sub>2</sub> external event counter input, and the MICROBUS™ option. All other options are available.

COP340, COP341, and COP342 are extended temperature versions of the COP440, COP441, and COP442, respectively.

## COP440 Series Instruction Set

Table 2 is a symbol table providing internal architecture, instruction operand and operation symbols used in the instruction set table.

Table 3 provides the mnemonic, operand, machine code, data flow, skip conditions and description associated with each instruction in the COP440 series instruction set.

**Table 2. COP440 Series Instruction Set Symbols**

Symbol	Definition	Symbol	Definition
INTERNAL ARCHITECTURE SYMBOLS		INSTRUCTION OPERAND SYMBOLS	
A	4-bit Accumulator	d	4-bit Operand Field, 0-15 binary (RAM Digit Select)
B	8-bit RAM Address Register	r	4-bit Operand Field, 0-9 binary (RAM Register Select)
Br	Upper 4 bits of B (register address)	a	11-bit Operand Field, 0-2047 binary (ROM Address)
Bd	Lower 4 bits of B (digit address)	y	4-bit Operand Field, 0-15 binary (Immediate Data)
C	1-bit Carry Register	RAM(s)	Content of RAM location addressed by s
D	4-bit Data Output Port	RAM <sub>N</sub>	Content of RAM location addressed by stack pointer N
EN	8-bit Enable Register	ROM(t)	Content of ROM location addressed by t
G	4-bit Register to latch data for G I/O Port	OPERATIONAL SYMBOLS	
H	4-bit Register to latch data for H I/O Port	+	Plus
IL	Two 1-bit Latches associated with the IN <sub>3</sub> or IN <sub>0</sub> Inputs	-	Minus
IN	4-bit Input Port	→	Replaces
IN <sub>7</sub> Z	Zero-Crossing Input	↔	Is exchanged with
L	8-bit TRI-STATE® I/O Port	=	Is equal to
M	4-bit contents of RAM Memory pointed to by B Register	Ā	The one's complement of A
N	2-bit subroutine return address stack pointer	⊕	Exclusive-OR
PC	11-bit ROM Address Register (program counter)	:	Range of values
Q	8-bit Register to latch data for L I/O Port	V	OR
R	8-bit Register to latch data for R TRI-STATE I/O Port		
SIO	4-bit Shift Register and Counter		
SK	Logic-Controlled Clock Output		
T	8-bit Binary Counter Register		



Table 3. COP440 Series Instruction Set

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
ARITHMETIC/LOGIC INSTRUCTIONS						
ASC		30	$\boxed{00110000}$	$A + C + RAM(B) \rightarrow A$ Carry $\rightarrow C$	Carry	Add with Carry, Skip on Carry
ADD		31	$\boxed{00110001}$	$A + RAM(B) \rightarrow A$	None	Add RAM to A
ADT		4A	$\boxed{01001010}$	$A + 10_{10} \rightarrow A$	None	Add Ten to A
AISC	y	5-	$\boxed{0101 \mid y}$	$A + y \rightarrow A$	Carry	Add Immediate, Skip on Carry (y $\neq$ 0)
CASC		10	$\boxed{00010000}$	$\bar{A} + RAM(B) + C \rightarrow A$ Carry $\rightarrow C$	Carry	Complement and Add with Carry, Skip on Carry
CLRA		00	$\boxed{00000000}$	$0 \rightarrow A$	None	Clear A
COMP		40	$\boxed{01000000}$	$\bar{A} \rightarrow A$	None	One's complement of A to A
NOP		44	$\boxed{01000100}$	None	None	No Operation
OR		33 1A	$\boxed{00110011}$ $\boxed{00011010}$	$A \vee M \rightarrow A$	None	OR RAM with A
RC		32	$\boxed{00110010}$	"0" $\rightarrow C$	None	Reset C
SC		22	$\boxed{00100010}$	"1" $\rightarrow C$	None	Set C
XOR		02	$\boxed{00000010}$	$A \oplus RAM(B) \rightarrow A$	None	Exclusive-OR RAM with A
TRANSFER OF CONTROL INSTRUCTIONS						
JID		FF	$\boxed{11111111}$	ROM (PC <sub>10:8</sub> , A, M) $\rightarrow$ PC <sub>7:0</sub>	None	Jump Indirect (Note 3)
JMP	a	6- --	$\boxed{01100 \mid a_{10:6}}$ $\boxed{a_{7:0}}$	$a \rightarrow PC$	None	Jump
JP	a	--	$\boxed{1 \mid a_{6:0}}$ (pages 2,3 only)	$a \rightarrow PC_{6:0}$	None	Jump within Page (Note 4)
		--	or $\boxed{11 \mid a_{5:0}}$ (all other pages)	$a \rightarrow PC_{5:0}$		
JSRP	a	--	$\boxed{10 \mid a_{5:0}}$	PC + 1 $\rightarrow$ RAM <sub>N</sub> N + 1 $\rightarrow$ N 00010 $\rightarrow$ PC <sub>10:6</sub> $a \rightarrow$ PC <sub>5:0</sub>	None	Jump to Subroutine Page (Note 5)
JSR	a	6- --	$\boxed{01101 \mid a_{10:6}}$ $\boxed{a_{7:0}}$	PC + 1 $\rightarrow$ RAM <sub>N</sub> N + 1 $\rightarrow$ N $a \rightarrow$ PC	None	Jump to Subroutine
RET		48	$\boxed{01001000}$	N - 1 $\rightarrow$ N RAM <sub>N</sub> $\rightarrow$ PC	None	Return from Subroutine
RETSK		49	$\boxed{01001001}$	N - 1 $\rightarrow$ N RAM <sub>N</sub> $\rightarrow$ PC	Always Skip on Return	Return from Subroutine then Skip

Table 3. COP440 Series Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description						
MEMORY REFERENCE INSTRUCTIONS												
CAME		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	A → EN <sub>7,4</sub>	None	Copy A, RAM to EN				
		0011	0011									
1F	<table border="1"><tr><td>0001</td><td>1111</td></tr></table>	0001	1111	RAM(B) → EN <sub>3,0</sub>								
0001	1111											
CAMQ		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	A → Q <sub>7,4</sub>	None	Copy A, RAM to Q				
		0011	0011									
3C	<table border="1"><tr><td>0011</td><td>1100</td></tr></table>	0011	1100	RAM(B) → Q <sub>3,0</sub>								
0011	1100											
CAME		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	A → T <sub>7,4</sub>	None	Copy A, RAM to T				
		0011	0011									
3F	<table border="1"><tr><td>0011</td><td>1111</td></tr></table>	0011	1111	RAM(B) → T <sub>3,0</sub>								
0011	1111											
CEMA		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	EN <sub>7,4</sub> → RAM(B)	None	Copy EN to RAM, A				
		0011	0011									
0F	<table border="1"><tr><td>0000</td><td>1111</td></tr></table>	0000	1111	EN <sub>3,0</sub> → A								
0000	1111											
CQMA		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	Q <sub>7,4</sub> → RAM(B)	None	Copy Q to RAM, A				
		0011	0011									
2C	<table border="1"><tr><td>0010</td><td>1100</td></tr></table>	0010	1100	Q <sub>3,0</sub> → A								
0010	1100											
CTMA		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	T <sub>7,4</sub> → RAM(B)	None	Copy T to RAM, A				
		0011	0011									
2F	<table border="1"><tr><td>0010</td><td>1111</td></tr></table>	0010	1111	T <sub>3,0</sub> → A								
0010	1111											
LD	r	-5	<table border="1"><tr><td>00</td><td>r</td><td>0101</td></tr><tr><td colspan="3">r = 0:3</td></tr></table>	00	r	0101	r = 0:3			RAM(B) → A Br ⊕ r → Br	None	Load RAM into A, Exclusive-OR Br with r
00	r	0101										
r = 0:3												
LDD	r,d	23	<table border="1"><tr><td>00</td><td>10</td><td>0011</td></tr></table>	00	10	0011	RAM(r,d) → A	None	Load A with RAM pointed to directly by r,d			
		00	10	0011								
--	<table border="1"><tr><td>0</td><td>r</td><td>d</td></tr><tr><td colspan="3">r = 0:7</td></tr></table>	0	r	d	r = 0:7							
0	r	d										
r = 0:7												
LID		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	ROM(PC <sub>10:8</sub> ,A,M) → M,A	None	Load RAM, A Indirect				
		0011	0011									
19	<table border="1"><tr><td>0001</td><td>1001</td></tr></table>	0001	1001									
0001	1001											
LQID		BF	<table border="1"><tr><td>1011</td><td>1111</td></tr></table>	1011	1111	ROM(PC <sub>10:8</sub> ,A,M) → Q	None	Load Q Indirect (Note 3)				
1011	1111											
RMB		0	4C	<table border="1"><tr><td>0100</td><td>1100</td></tr></table>	0100	1100	0 → RAM(B) <sub>0</sub>	None	Reset RAM Bit			
		0100	1100									
		1	45	<table border="1"><tr><td>0100</td><td>0101</td></tr></table>	0100	0101	0 → RAM(B) <sub>1</sub>					
		0100	0101									
2	42	<table border="1"><tr><td>0100</td><td>0010</td></tr></table>	0100	0010	0 → RAM(B) <sub>2</sub>							
0100	0010											
3	43	<table border="1"><tr><td>0100</td><td>0011</td></tr></table>	0100	0011	0 → RAM(B) <sub>3</sub>							
0100	0011											
SMB		0	4D	<table border="1"><tr><td>0100</td><td>1101</td></tr></table>	0100	1101	1 → RAM(B) <sub>0</sub>	None	Set RAM Bit			
		0100	1101									
		1	47	<table border="1"><tr><td>0100</td><td>0111</td></tr></table>	0100	0111	1 → RAM(B) <sub>1</sub>					
		0100	0111									
2	46	<table border="1"><tr><td>0100</td><td>0110</td></tr></table>	0100	0110	1 → RAM(B) <sub>2</sub>							
0100	0110											
3	4B	<table border="1"><tr><td>0100</td><td>1011</td></tr></table>	0100	1011	1 → RAM(B) <sub>3</sub>							
0100	1011											
STII	y	7-	<table border="1"><tr><td>0111</td><td>y</td></tr></table>	0111	y	y → RAM(B) Bd + 1 → Bd	None	Store Memory Immediate and Increment Bd				
0111	y											
X	r	-6	<table border="1"><tr><td>00</td><td>r</td><td>0110</td></tr><tr><td colspan="3">r = 0:3</td></tr></table>	00	r	0110	r = 0:3			RAM(B) ↔ A Br ⊕ r → Br	None	Exchange RAM with A, Exclusive-OR Br with r
00	r	0110										
r = 0:3												
XAD	r,d	23	<table border="1"><tr><td>0010</td><td>0011</td></tr></table>	0010	0011	RAM(r,d) ↔ A	None	Exchange A with RAM pointed to directly by r,d				
		0010	0011									
--	<table border="1"><tr><td>1</td><td>r</td><td>d</td></tr><tr><td colspan="3">r = 0:7</td></tr></table>	1	r	d	r = 0:7							
1	r	d										
r = 0:7												
XDS	r	-7	<table border="1"><tr><td>00</td><td>r</td><td>0111</td></tr><tr><td colspan="3">r = 0:3</td></tr></table>	00	r	0111	r = 0:3			RAM(B) ↔ A Bd - 1 → Bd Br ⊕ r → Br	Bd decrements past 0	Exchange RAM with A and Decrement Bd, Exclusive-OR Br with r
00	r	0111										
r = 0:3												
XIS	r	-4	<table border="1"><tr><td>00</td><td>r</td><td>0100</td></tr><tr><td colspan="3">r = 0:3</td></tr></table>	00	r	0100	r = 0:3			RAM(B) ↔ A Bd + 1 → Bd Br ⊕ r → Br	Bd increments past 15	Exchange RAM with A and Increment Bd, Exclusive-OR Br with r
00	r	0100										
r = 0:3												

Table 3. COP440 Series Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description																																
<b>REGISTER REFERENCE INSTRUCTIONS</b>																																						
CAB		50	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	1	0	1	0	0	0	0	A → Bd	None	Copy A to Bd																								
0	1	0	1	0	0	0	0																															
CBA		4E	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td></tr></table>	0	1	0	0	1	1	1	0	Bd → A	None	Copy Bd to A																								
0	1	0	0	1	1	1	0																															
LBI	r,d	--	<table border="1"><tr><td>0</td><td>0</td><td>r</td><td> </td><td>d</td><td>-</td><td>1</td></tr></table> r=0:3,d=0,9:15 or <table border="1"><tr><td>1</td><td> </td><td>r</td><td> </td><td>d</td></tr></table> r=0:7,any d	0	0	r		d	-	1	1		r		d	r,d → B	Skip until not a LBI	Load B Immediate with r,d (Note 6)																				
0	0	r		d	-	1																																
1		r		d																																		
LEI	y	33 6-	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table> <table border="1"><tr><td>0</td><td>1</td><td>1</td><td>0</td><td> </td><td>y</td></tr></table>	0	0	1	1	0	0	1	1	0	1	1	0		y	y → EN <sub>3:0</sub>	None	Load lower half of EN Immediate																		
0	0	1	1	0	0	1	1																															
0	1	1	0		y																																	
XABR		12	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table>	0	0	0	1	0	0	1	0	A ↔ Br	None	Exchange A with Br																								
0	0	0	1	0	0	1	0																															
XAN		33 0B	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table> <table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	1	1	0	0	1	1	0	0	0	0	1	0	1	1	A ↔ N(0,0 → A <sub>3</sub> ,A <sub>2</sub> )	None	Exchange A with N																
0	0	1	1	0	0	1	1																															
0	0	0	0	1	0	1	1																															
<b>TEST INSTRUCTIONS</b>																																						
SKC		20	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	1	0	0	0	0	0		C = "1"	Skip if C is True																								
0	0	1	0	0	0	0	0																															
SKE		21	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	1	0	0	0	0	1		A = RAM(B)	Skip if A Equals RAM																								
0	0	1	0	0	0	0	1																															
SKGZ		33 21	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table> <table border="1"><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	1	1	0	0	1	1	0	0	1	0	0	0	0	1		G <sub>3:0</sub> = 0	Skip if G is Zero (all 4 bits)																
0	0	1	1	0	0	1	1																															
0	0	1	0	0	0	0	1																															
SKGBZ		33	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	1	1	0	0	1	1	1st byte		Skip if G Bit is Zero																								
0	0	1	1	0	0	1	1																															
	0	01	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	0	0	0	0	0	1	} 2nd byte	G <sub>0</sub> = 0																									
0	0	0	0	0	0	0	1																															
	1	11	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	0	1	0	0	0	1		G <sub>1</sub> = 0																									
0	0	0	1	0	0	0	1																															
	2	03	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	0	0	0	0	1	1	G <sub>2</sub> = 0																										
0	0	0	0	0	0	1	1																															
	3	13	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	0	1	0	0	1	1	G <sub>3</sub> = 0																										
0	0	0	1	0	0	1	1																															
SKMBZ		0 1 2 3	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table> <table border="1"><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table> <table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table> <table border="1"><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	1	1		RAM(B) <sub>0</sub> = 0 RAM(B) <sub>1</sub> = 0 RAM(B) <sub>2</sub> = 0 RAM(B) <sub>3</sub> = 0	Skip if RAM Bit is Zero
0	0	0	0	0	0	0	1																															
0	0	0	1	0	0	0	1																															
0	0	0	0	0	0	1	1																															
0	0	0	1	0	0	1	1																															
SKSZ		33 1C	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table> <table border="1"><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td></tr></table>	0	0	1	1	0	0	1	1	0	0	0	1	1	1	0	0		SIO = 0	Skip if SIO is Zero																
0	0	1	1	0	0	1	1																															
0	0	0	1	1	1	0	0																															
SKT		41	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	1	0	0	0	0	0	1		T counter carry has occurred since last test	Skip on Timer (Note 3)																								
0	1	0	0	0	0	0	1																															

Table 3. COP440 Series Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
INPUT/OUTPUT INSTRUCTIONS						
CAMR		33	0011 0011	A → R <sub>7:4</sub>	None	Output A, RAM to R Port
		3D	0011 1101	RAM(B) → R <sub>3:0</sub>		
ING		33	0011 0011	G → A	None	Input G Port to A
		2A	0010 1010			
INH		33	0011 0011	H → A	None	Input H Port to A
		2B	0010 1011			
ININ		33	0011 0011	IN → A	None	Input IN Inputs to A (Note 2)
		28	0010 1000			
INIL		33	0011 0011	IL <sub>3</sub> , CKO, IN <sub>1:Z</sub> , IL <sub>0</sub> → A	None	Input IL Latches to A (Note 3)
		29	0010 1001			
INL		33	0011 0011	L <sub>7:4</sub> → RAM(B)	None	Input L Port to RAM, A
		2E	0010 1110	L <sub>3:0</sub> → A		
INR		33	0011 0011	R <sub>7:4</sub> → RAM(B)	None	Input R Port to RAM, A
		2D	0010 1101	R <sub>3:0</sub> → A		
OBD		33	0011 0011	Bd → D	None	Output Bd to D Port
		3E	0011 1110			
OGI	y	33	0011 0011	y → G	None	Output to G Port Immediate
		5-	0101  y			
OMG		33	0011 0011	RAM(B) → G	None	Output RAM to G Port
		3A	0011 1010			
OMH		33	0011 0011	RAM(B) → H	None	Output RAM to H Port
		3B	0011 1011			
XAS		4F	0100 1111	A ↔ SIO, C → SKL	None	Exchange A with SIO (Note 3)

**Note 1:** All subscripts for alphabetical symbols indicate bit numbers unless explicitly defined (e.g., Br and Bd are explicitly defined). Bits are numbered 0 to N where 0 signifies the least significant bit (low-order, right-most bit). For example, A<sub>3</sub> indicates the most significant (left-most) bit of the 4-bit A register.

**Note 2:** The ININ instruction is not available on the 24-pin COP442/COP342 since this device does not contain the IN inputs.

**Note 3:** For additional information on the operation of the XAS, JID, LQID, INIL, and SKT instructions, see below.

**Note 4:** The JP instruction allows a jump, while in subroutine pages 2 or 3, to any ROM location within the two-page boundary of pages 2 or 3. The JP instruction, otherwise, permits a jump to a ROM location within the current 64-word page. JP may not jump to the last word of a page.

**Note 5:** A JSRP transfers program control to subroutine page 2 (00010 is loaded into the upper 5 bits of P). A JSRP may not be used when in pages 2 or 3. JSRP may not jump to the last word in page 2.

**Note 6:** LBI is a single-byte instruction if d = 0, 9, 10, 11, 12, 13, 14, or 15. The machine code for the lower 4 bits equals the binary value of the "d" data minus 1, e.g., to load the lower four bits of B (Bd) with the value 9 (1001<sub>2</sub>), the lower 4 bits of the LBI instruction equal 8 (1000<sub>2</sub>). To load 0, the lower 4 bits of the LBI instruction should equal 15 (1111<sub>2</sub>).

## Description of Selected Instructions

The following information is provided to assist the user in understanding the operation of several unique instructions and to provide notes useful to programmers in writing COP440 programs.

### XAS Instruction

XAS (Exchange A with SIO) exchanges the 4-bit contents of the accumulator with the 4-bit contents of the SIO register. The contents of SIO will contain serial-in/serial-out shift register or binary counter data, depending on the value of the EN register. An XAS instruction will also affect the SK output. (See Functional Description, EN register, above). If SIO is selected as a shift register, an XAS instruction must be performed once every 4 instruction cycles to effect a continuous data stream.

### JID Instruction

JID (Jump Indirect) is an indirect addressing instruction, transferring program control to a new ROM location pointed to indirectly by A and M. It loads the lower 8 bits of the ROM address register PC with the contents of ROM addressed by the 11-bit word, PC<sub>10:8</sub>, A, M. PC<sub>10</sub>, PC<sub>9</sub> and PC<sub>8</sub> are not affected by this instruction.

Note that JID requires 2 instruction cycles if executed, 1 instruction cycle time if skipped.

### INIL Instruction

INIL (Input IL Latches to A) inputs 2 latches, IL<sub>3</sub> and IL<sub>0</sub>, CKO and IN<sub>1</sub> into A (see Figure 13). The IL<sub>3</sub> and IL<sub>0</sub> latches are set if a low-going pulse ("1" to "0") has occurred on the IN<sub>3</sub> and IN<sub>0</sub> inputs since the last INIL instruction, provided the input pulse stays low for at least two instruction cycles. Execution of an INIL inputs IL<sub>3</sub> and IL<sub>0</sub> into A3 and A0 respectively, and resets these latches to allow them to respond to subsequent low-going pulses on the IN<sub>3</sub> and IN<sub>0</sub> lines. If CKO is mask-programmed as a general purpose input, an INIL will input the state of CKO into A2. If CKO has not been so programmed, a "1" will be placed in A2. Unlike the COP420/420C/420L/444L series, INIL will input IN<sub>1</sub> into A1. If zero-crossing detect is selected, the IN<sub>1</sub> input will go through the detection logic, thus allowing the user to interrogate the input,

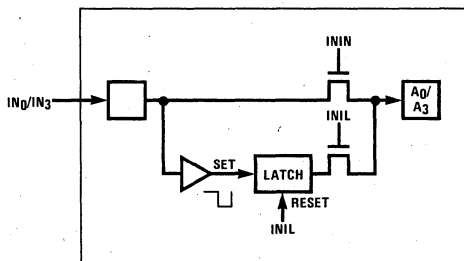


Figure 13. INIL Hardware Implementation

sending a "1" if the input is above zero volts and a "0" if it is below zero volts. INIL is useful in recognizing pulses of short duration or pulses which occur too often to be read conveniently by an ININ instruction. It is also useful in checking the status of the zero-crossing detect input. The general purpose input IN<sub>3</sub>-IN<sub>0</sub> are input to A upon execution of an ININ instruction, and the IN<sub>1</sub> input does not go through zero-crossing logic so that it has the same logic level as the other IN inputs for the ININ instruction (see Figure 9).

Note: IL latches are cleared on reset. This is different from the COP420/420C/420L/444L series.

### LQID Instruction

LQID (Load Q Indirect) loads the 8-bit Q register with the contents of ROM pointed to by the 11-bit word PC<sub>10:PC<sub>8</sub></sub>, A, M. LQID can be used for table lookup or code conversion such as BCD to seven-segment. Note that LQID takes two instruction cycles if executed and one instruction cycle if skipped. Unlike most other COPSTM processors, this instruction does not push the stack.

### LID Instruction

LID (Load Indirect) loads M and A with the contents of ROM pointed to by the 11-bit word PC<sub>10:PC<sub>8</sub></sub>, A, M. Note that LID takes three instruction cycles if executed and two if skipped.

### SKT Instruction

The SKT (Skip On Timer) instruction tests the state of the T counter (see internal logic, above) overflow latch, executing the next program instruction if the latch is not set. If the latch has been set since the previous test, the next program instruction is skipped and the latch is reset. The features associated with this instruction allow the processor to generate its own time-base for real-time processing, rather than relying on an external input signal.

### Instruction Set Notes

- The first word of a COP440 program (ROM address 0) must be a CLRA (Clear A) instruction.
- Although skipped instructions are not executed, they are still fetched from program memory. Thus program paths take the same number of cycle times whether instructions are skipped or executed, except for LID, LQID, and JID.
- The ROM is organized into 32 pages of 64 words each. The Program Counter is an 11-bit binary counter, and will count through page boundaries. If a JP, JSRP, JID, LQID, or LID instruction is the last word of a page, the instruction operates as if it were in the next page. For example: a JP located in the last word of a page will jump to a location in the next page. Also, a LQID or JID located in the last word of page 3, 7, 11, 15, 19, 23, 27, or 31 will access data in the next group of four pages.

## Option List

The COP440 mask-programmable options are assigned numbers which correspond with the COP440 pins.

### Option 1: L<sub>1</sub> I/O Port (see note below)

- = 0: Standard output
- = 1: Open-drain output
- = 2: LED direct drive output
- = 3: TRI-STATE<sup>®</sup> output
- = 4: same as 0 with extra load device to V<sub>CC</sub>
- = 5: same as 1 with extra load device to V<sub>CC</sub>
- = 6: same as 2 with extra load device to V<sub>CC</sub>
- = 7: same as 3 with extra load device to V<sub>CC</sub>

### Option 2: L<sub>0</sub> I/O Port

(same as Option 1)

### Option 3: SI Input

- = 0: Input with load device to V<sub>CC</sub>
- = 1: Hi-Z input

### Option 4: SO Output

- = 0: Standard output
- = 1: Open-drain output
- = 2: Push-pull output

### Option 5: SK Output

(same as Option 4)

### Option 6: IN<sub>0</sub> Input

(same as Option 3)

### Option 7: IN<sub>3</sub> Input

(same as Option 3)

### Option 8: G<sub>0</sub> I/O Port

- = 0: Standard output
- = 1: Open-drain output

### Option 9: G<sub>1</sub> I/O Port

(same as Option 8)

### Option 10: G<sub>2</sub> I/O Port

(same as Option 8)

### Option 11: G<sub>3</sub> I/O Port

(same as Option 8)

### Option 12: H<sub>0</sub> I/O Port

(same as Option 8)

### Option 13: H<sub>1</sub> I/O Port

(same as Option 8)

### Option 14: H<sub>2</sub> I/O Port

(same as Option 8)

### Option 15: H<sub>3</sub> I/O Port

(same as Option 8)

### Option 16: D<sub>3</sub> Output

(same as Option 8)

### Option 17: D<sub>2</sub> Output

(same as Option 8)

### Option 18: D<sub>1</sub> Output

(same as Option 8)

### Option 19: D<sub>0</sub> Output

(same as Option 8)

### Option 20: GND — No options available

### Option 21: CKO Pin

- = 0: Oscillator output
- = 1: RAM power supply (V<sub>R</sub>) input
- = 2: General purpose input with load device to V<sub>CC</sub>
- = 3: General purpose Hi-Z input

### Option 22: CKI Input

- = 0: Crystal input divided by 16
- = 1: Crystal input divided by 8
- = 2: Single-pin RC controlled oscillator (+ 4)
- = 3: Schmitt trigger clock input (+ 4)

### Option 23: $\overline{\text{RESET}}$ Input

(same as Option 3)

### Option 24: R<sub>7</sub> I/O Port (see note below)

- = 0: Standard output
- = 1: Open-drain output
- = 2: Push-pull output
- = 3: TRI-STATE<sup>®</sup> output
- = 4: same as 0 with extra load device to V<sub>CC</sub>
- = 5: same as 1 with extra load device to V<sub>CC</sub>
- = 6: same as 2 with extra load device to V<sub>CC</sub>
- = 7: same as 3 with extra load device to V<sub>CC</sub>

### Option 25: R<sub>6</sub> I/O Port

(same as Option 24)

### Option 26: R<sub>5</sub> I/O Port

(same as Option 24)

### Option 27: R<sub>4</sub> I/O Port

(same as Option 24)

### Option 28: R<sub>3</sub> I/O Port

(same as Option 24)

### Option 29: R<sub>2</sub> I/O Port

(same as Option 24)

### Option 30: R<sub>1</sub> I/O Port

(same as Option 24)

### Option 31: R<sub>0</sub> I/O Port

(same as Option 24)

### Option 32: L<sub>7</sub> I/O Port

(same as Option 1)

### Option 33: L<sub>6</sub> I/O Port

(same as Option 1)

### Option 34: L<sub>5</sub> I/O Port

(same as Option 1)

### Option 35: L<sub>4</sub> I/O Port

(same as Option 1)

### Option 36: IN<sub>1</sub> Input

- = 0: Input with load device to V<sub>CC</sub>
- = 1: Hi-Z Input
- = 2: Zero-crossing detect input (Option 41 = 0)

### Option 37: IN<sub>2</sub> Input

(same as Option 3)

### Option 38: L<sub>3</sub> I/O Port

(same as Option 1)

**Option List (continued)**

Option 39: L<sub>2</sub> I/O Port  
(same as Option 1)

Option 40: V<sub>CC</sub> — no options available

Option 41: COP Function  
= 0: Normal  
= 1: MICROBUS™ option

Option 42: IN Input Levels  
= 0: Standard TTL input levels ("0" = 0.8V, "1" = 2.0V)  
= 1: Higher voltage input levels ("0" = 1.2V, "1" = 3.6V)

Option 43: G Input Levels  
(same as Option 42)

Option 44: L Input Levels  
(same as Option 42)

Option 45: CKO Input Levels  
(same as Option 42)

Option 46: SI Input Levels  
(same as Option 42)

Option 47: R Input Levels  
(same as Option 42)

Option 48: H Input Levels  
(same as Option 42)

Option 49: No option available

Option 50: COP Bonding  
= 0: COP440 (40-pin device)  
= 1: COP441 (28-pin device)  
= 2: COP442 (24-pin device)  
= 3: COP440 and COP441  
= 4: COP440 and COP442  
= 5: COP440, COP441, and COP442  
= 6: COP441 and COP442

**Note on L and R I/O Port Options**

If L and R I/O Ports are used as inputs, the following must be observed:

- a. Open-Drain output (selection 1) is allowed only if external pull-up is provided.
- b. If L and R output ports are disabled when reading, an external pull-up is required unless selections 4, 5, 6, or 7 are chosen.
- c. If L output port is enabled, selections 3 and 7 are not allowed.
- d. If R output port is enabled, selections 2, 3, 6, and 7 are not allowed.

**Test Mode (Non-Standard Operation)**

The SO output has been configured to provide for standard test procedures for the custom-programmed COP440. With SO forced to logic "1", two test modes are provided, depending upon the value of SI:

- a. RAM and Internal Logic Test Mode (SI = 1)
- b. ROM Test Mode (SI = 0)

These special test modes should not be employed by the user; they are intended for manufacturing test only.

# COP444C/COP445C and COP344C/COP345C Single-Chip CMOS Microcontrollers

## General Description

The COP444C, COP445C, COP344C, and COP345C Single-Chip CMOS Microcontrollers are members of the COPS™ family, fabricated using double-poly, silicon-gate CMOS technology. These controller-oriented processors are complete microcomputers containing all system timing, internal logic, ROM, RAM, and I/O necessary to implement dedicated control functions in a variety of applications. Features include single supply operation, a variety of output configuration options, with an instruction set, internal architecture and I/O scheme designed to facilitate keyboard input, display output and BCD data manipulation. The COP445C is identical to the COP444C, but with 19 I/O lines instead of 23. They are an appropriate choice for use in numerous human interface control environments. Standard test procedures and reliable high-density fabrication techniques provide the medium to large volume customers with a customized controller oriented processor at a low end-product cost.

The COP344C and COP345C are exact functional equivalents, but extended temperature range versions of the COP444C and COP445C respectively.

COPS, MICROWIRE, and MICROBUS are trademarks of National Semiconductor Corp. TRI-STATE is a registered trademark of National Semiconductor Corp.

## Features

- Lowest power dissipation (50μW typical)
- Power-saving IDLE state and HALT mode
- Powerful instruction set
- 2k x 8 ROM, 128 x 4 RAM
- 23 I/O lines (COP444C)
- True vectored interrupt, plus restart
- Three-level subroutine stack
- 4μs instruction time, plus software selectable oscillators
- Single supply operation (2.4-5.5V)
- Programmable time-base counter for real-time processing
- Internal binary counter register with MICROWIRE™ serial I/O capability
- General purpose and TRI-STATE® outputs
- LSTTL/CMOS compatible
- MICROBUS™ compatible
- Software/hardware compatible with other members of COP400 family
- Extended temperature range devices COP344C/COP345C (-40°C to +85°C)

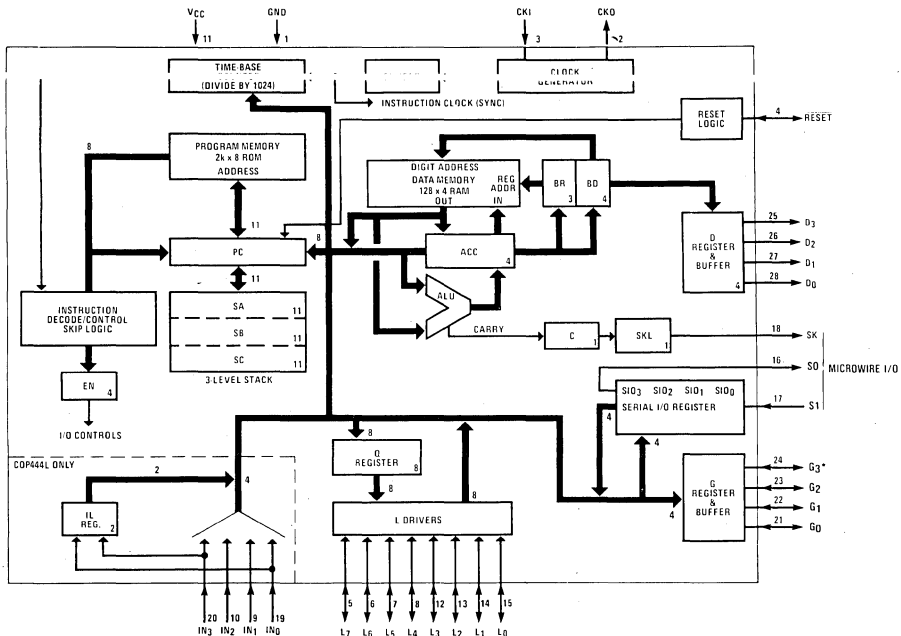


Figure 1. COP344C/COP345C, COP444C/COP445C Block Diagram





# COP444L/COP445L and COP344L/COP345L Single-Chip N-Channel Microcontrollers

## General Description

The COP444L, COP445L, COP344L, and COP345L Single-Chip N-Channel Microcontrollers are members of the COPS™ family, fabricated using N-channel, silicon gate MOS technology. These controller oriented processors are complete microcomputers containing all system timing, internal logic, ROM, RAM, and I/O necessary to implement dedicated control functions in a variety of applications. Features include single supply operation, a variety of output configuration options, with an instruction set, internal architecture and I/O scheme designed to facilitate keyboard input, display output and BCD data manipulation. The COP445L is identical to the COP444L, but with 19 I/O lines instead of 23. They are an appropriate choice for use in numerous human interface control environments. Standard test procedures and reliable high-density fabrication techniques provide the medium to large volume customers with a customized controller oriented processor at a low end-product cost.

The COP344L and COP345L are exact functional equivalents, but extended temperature range versions of the COP444L and COP445L respectively.

COPS and MICROWIRE are trademarks of National Semiconductor Corp. TRI-STATE is a registered trademark of National Semiconductor Corp.

## Features

- Low cost
- Powerful instruction set
- 2k x 8 ROM, 128 x 4 RAM
- 23 I/O lines (COP444L)
- True vectored interrupt, plus restart
- Three-level subroutine stack
- 15µs instruction time
- Single supply operation (4.5-6.3V)
- Low current drain (11mA max.)
- Internal time-base counter for real-time processing
- Internal binary counter register with MICROWIRE™ serial I/O capability
- General purpose and TRI-STATE® outputs
- LSTTL/CMOS compatible in and out
- Direct drive of LED digit and segment lines
- Software/hardware compatible with other members of COP400 family
- Extended temperature range devices COP344L/COP345L (-40°C to +85°C)
- Wider supply range (4.5-9.5V) optionally available

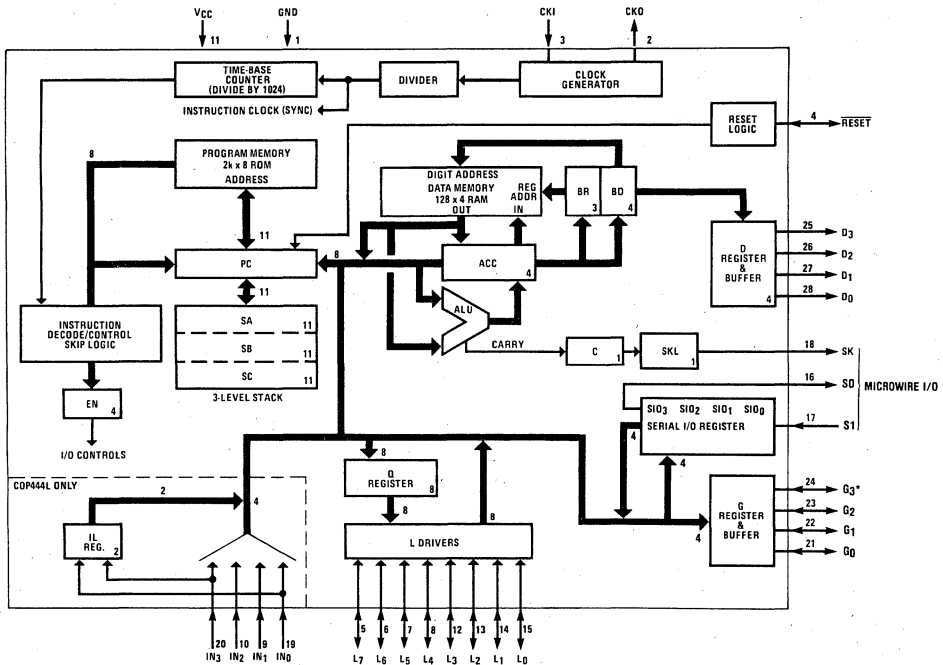


Figure 1. COP344L/COP345L, COP444L/COP445L Block Diagram

**COP444L/COP445L****Absolute Maximum Ratings**

Voltage at Any Pin Relative to GND	-0.5V to +10V
Ambient Operating Temperature	0°C to +70°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C
Power Dissipation	0.75 Watt at 25°C 0.4 Watt at 70°C
Total Source Current	120 mA
Total Sink Current	120 mA

*Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

**DC Electrical Characteristics** 0°C ≤ T<sub>A</sub> ≤ +70°C, 4.5V ≤ V<sub>CC</sub> ≤ 9.5V unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Standard Operating Voltage (V <sub>CC</sub> )	Note 1	4.5	6.3	V
Optional Operating Voltage (V <sub>CC</sub> )		4.5	9.5	V
Power Supply Ripple	peak to peak		0.5	V
Operating Supply Current	all inputs and outputs open		13	mA
Input Voltage Levels				
CKI Input Levels				
Crystal Input (±32, ±16, ±8)				
Logic High (V <sub>IH</sub> )		2.0		V
Logic Low (V <sub>IL</sub> )		-0.3	0.4	V
Schmitt Trigger Input (±4)				
Logic High (V <sub>IH</sub> )		0.7V <sub>CC</sub>		V
Logic Low (V <sub>IL</sub> )		-0.3	0.6	V
RESET Input Levels	Schmitt trigger input			
Logic High		0.7V <sub>CC</sub>		V
Logic Low		-0.3	0.6	V
SU Input Level (test mode)		2.0	2.5	V
All Other Inputs				
Logic High	V <sub>CC</sub> = Max.	3.0		V
Logic High	with TTL trip level options	2.0		V
Logic Low	selected, V <sub>CC</sub> = 5V ± 5%	-0.3	0.8	V
Logic High	with high trip level options	3.6		V
Logic Low	selected	-0.3	1.2	V
Input Capacitance			7	pF
Hi-Z Input Leakage		-1	+1	μA
Output Voltage Levels				
LSTTL Operation	V <sub>CC</sub> = 5V ± 5%			
Logic High (V <sub>OH</sub> )	I <sub>OH</sub> = -25 μA	2.7		V
Logic Low (V <sub>OL</sub> )	I <sub>OL</sub> = 0.36 mA		0.4	V
CMOS Operation				
Logic High	I <sub>OH</sub> = -10 μA	V <sub>CC</sub> - 1		V
Logic Low	I <sub>OL</sub> = +10 μA		0.2	V

**Note 1:** V<sub>CC</sub> voltage change must be less than 0.5V in a 1ms period to maintain proper operation.

**COP444L/COP445L****DC Electrical Characteristics** (continued)  $0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 9.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
<b>Output Current Levels</b>				
<b>Output Sink Current</b>				
SO and SK Outputs ( $I_{OL}$ )	$V_{CC} = 9.5\text{V}$ , $V_{OL} = 0.4\text{V}$	1.8		mA
	$V_{CC} = 6.3\text{V}$ , $V_{OL} = 0.4\text{V}$	1.2		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.9		mA
L <sub>0</sub> -L <sub>7</sub> Outputs and Standard	$V_{CC} = 9.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.8		mA
G <sub>0</sub> -G <sub>3</sub> , D <sub>0</sub> -D <sub>3</sub> Outputs ( $I_{OL}$ )	$V_{CC} = 6.3\text{V}$ , $V_{OL} = 0.4\text{V}$	0.5		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.4		mA
G <sub>0</sub> -G <sub>3</sub> and D <sub>0</sub> -D <sub>3</sub> Outputs with High Current Options ( $I_{OL}$ )	$V_{CC} = 9.5\text{V}$ , $V_{OL} = 1.0\text{V}$	15		mA
	$V_{CC} = 6.3\text{V}$ , $V_{OL} = 1.0\text{V}$	11		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 1.0\text{V}$	7.5		mA
G <sub>0</sub> -G <sub>3</sub> and D <sub>0</sub> -D <sub>3</sub> Outputs with Very High Current Options ( $I_{OL}$ )	$V_{CC} = 9.5\text{V}$ , $V_{OL} = 1.0\text{V}$	30		mA
	$V_{CC} = 6.3\text{V}$ , $V_{OL} = 1.0\text{V}$	22		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 1.0\text{V}$	15		mA
CKI (Single-pin RC oscillator)	$V_{CC} = 4.5\text{V}$ , $V_{IH} = 3.5\text{V}$	2		mA
CKO	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.2		mA
<b>Output Source Current</b>				
<b>Standard Configuration, All Outputs (<math>I_{OH}</math>)</b>				
	$V_{CC} = 9.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-140	-800	$\mu\text{A}$
	$V_{CC} = 6.3\text{V}$ , $V_{OH} = 2.0\text{V}$	-75	-480	$\mu\text{A}$
	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-30	-250	$\mu\text{A}$
<b>Push-Pull Configuration SO and SK Outputs (<math>I_{OH}</math>)</b>				
	$V_{CC} = 9.5\text{V}$ , $V_{OH} = 4.75\text{V}$	-1.4		mA
	$V_{CC} = 6.3\text{V}$ , $V_{OH} = 2.4\text{V}$	-1.4		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 1.0\text{V}$	-1.2		mA
<b>LED Configuration, L<sub>0</sub>-L<sub>7</sub> Outputs, Low Current Driver Option (<math>I_{OH}</math>)</b>				
	$V_{CC} = 9.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-1.5	-18	mA
	$V_{CC} = 6.0\text{V}$ , $V_{OH} = 2.0\text{V}$	-1.5	-13	mA
<b>LED Configuration, L<sub>0</sub>-L<sub>7</sub> Outputs, High Current Driver Option (<math>I_{OH}</math>)</b>				
	$V_{CC} = 9.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-3.0	-35	mA
	$V_{CC} = 6.0\text{V}$ , $V_{OH} = 2.0\text{V}$	-3.0	-25	mA
<b>TRI-STATE<sup>®</sup> Configuration, L<sub>0</sub>-L<sub>7</sub> Outputs, Low Current Driver Option (<math>I_{OH}</math>)</b>				
	$V_{CC} = 9.5\text{V}$ , $V_{OH} = 5.5\text{V}$	-0.75		mA
	$V_{CC} = 6.3\text{V}$ , $V_{OH} = 3.2\text{V}$	-0.8		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 1.5\text{V}$	-0.9		mA
<b>TRI-STATE<sup>®</sup> Configuration, L<sub>0</sub>-L<sub>7</sub> Outputs, High Current Driver Option (<math>I_{OH}</math>)</b>				
	$V_{CC} = 9.5\text{V}$ , $V_{OH} = 5.5\text{V}$	-1.5		mA
	$V_{CC} = 6.3\text{V}$ , $V_{OH} = 3.2\text{V}$	-1.6		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 1.5\text{V}$	-1.8		mA
Input Load Source Current	$V_{CC} = 5.0\text{V}$ , $V_{IL} = 0\text{V}$	-10	-140	$\mu\text{A}$
<b>CKO Output</b>				
RAM Power Supply Option Power Requirement	$V_R = 3.3\text{V}$		3.0	mA
TRI-STATE <sup>®</sup> Output Leakage Current		-2.5	+2.5	$\mu\text{A}$
<b>Total Sink Current Allowed</b>				
All Outputs Combined			120	mA
D, G Ports			120	mA
L <sub>7</sub> -L <sub>4</sub>			4	mA
L <sub>3</sub> -L <sub>0</sub>			4	mA
All Other Pins			1.5	mA
<b>Total Source Current Allowed</b>				
All I/O Combined			120	mA
L <sub>7</sub> -L <sub>4</sub>			60	mA
L <sub>3</sub> -L <sub>0</sub>			60	mA
Each L Pin			30	mA
All Other Pins			1.5	mA

**COP344L/COP345L****Absolute Maximum Ratings**

Voltage at Any Pin Relative to GND	-0.5V to +10V
Ambient Operating Temperature	-40°C to +85°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C
Power Dissipation	0.75 Watt at 25°C 0.25 Watt at 85°C
Total Source Current	120 mA
Total Sink Current	120 mA

*Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

**DC Electrical Characteristics**  $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 7.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Standard Operating Voltage ( $V_{CC}$ )	Note 1	4.5	5.5	V
Optional Operating Voltage ( $V_{CC}$ )		4.5	7.5	V
Power Supply Ripple	peak to peak		0.5	V
Operating Supply Current	all inputs and outputs open		15	mA
Input Voltage Levels				
CKI Input Levels				
Crystal Input				
Logic High ( $V_{IH}$ )		2.2		V
Logic Low ( $V_{IL}$ )		-0.3	0.3	V
Schmitt Trigger Input				
Logic High ( $V_{IH}$ )		$0.7 V_{CC}$		V
Logic Low ( $V_{IL}$ )		-0.3	0.4	V
RESET Input Levels	Schmitt Trigger Input			
Logic High		$0.7 V_{CC}$		V
Logic Low		-0.3	0.4	V
SO Input Level (Test mode)		2.2	2.5	V
All Other Inputs				
Logic High	$V_{CC} = \text{Max.}$	3.0		V
Logic High	with TTL trip level options	2.2		V
Logic Low	selected, $V_{CC} = 5V \pm 5\%$	-0.3	0.6	V
Logic High	with high trip level options	3.6		V
Logic Low	selected	-0.3	1.2	V
Input Capacitance			7	pF
Hi-Z Input Leakage		-2	+2	$\mu\text{A}$
Output Voltage Levels				
LSTTL Operation	$V_{CC} = 5V \pm 5\%$			
Logic High ( $V_{OH}$ )	$I_{OH} = -20\mu\text{A}$	2.7		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 0.36\text{mA}$		0.4	V
CMOS Operation				
Logic High	$I_{OH} = -10\mu\text{A}$	$V_{CC} - 1$		V
Logic Low	$I_{OL} = +10\mu\text{A}$		0.2	V

**Note 1:**  $V_{CC}$  voltage change must be less than 0.5V in a 1ms period to maintain proper operation.

## COP344L/COP345L

DC Electrical Characteristics (continued)  $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 7.5\text{V}$  unless otherwise noted.

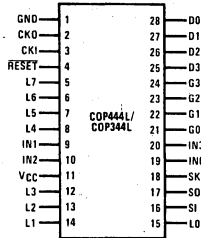
Parameter	Conditions	Min.	Max.	Units
<b>Output Current Levels</b>				
<b>Output Sink Current</b>				
SO and SK Outputs ( $I_{OL}$ )	$V_{CC} = 7.5\text{V}$ , $V_{OL} = 0.4\text{V}$	1.4		mA
	$V_{CC} = 5.5\text{V}$ , $V_{OL} = 0.4\text{V}$	1.0		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.8		mA
$L_0$ - $L_7$ Outputs, and Standard $G_0$ - $G_3$ , $D_0$ - $D_3$ Outputs ( $I_{OL}$ )	$V_{CC} = 7.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.6		mA
	$V_{CC} = 5.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.5		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.4		mA
$G_0$ - $G_3$ and $D_0$ - $D_3$ Outputs with High Current Options ( $I_{OL}$ )	$V_{CC} = 7.5\text{V}$ , $V_{OL} = 1.0\text{V}$	12		mA
	$V_{CC} = 5.5\text{V}$ , $V_{OL} = 1.0\text{V}$	9		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 1.0\text{V}$	7		mA
$G_0$ - $G_3$ and $D_0$ - $D_3$ Outputs with Very High Current Options ( $I_{OL}$ )	$V_{CC} = 7.5\text{V}$ , $V_{OL} = 1.0\text{V}$	24		mA
	$V_{CC} = 5.5\text{V}$ , $V_{OL} = 1.0\text{V}$	18		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 1.0\text{V}$	14		mA
CKI (Single-pin RC oscillator) CKO	$V_{CC} = 4.5\text{V}$ , $V_{IH} = 3.5\text{V}$	2		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.2		mA
<b>Output Source Current</b>				
<b>Standard Configuration, All Outputs (<math>I_{OH}</math>)</b>				
	$V_{CC} = 7.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-100	-900	$\mu\text{A}$
	$V_{CC} = 5.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-55	-600	$\mu\text{A}$
	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-28	-350	$\mu\text{A}$
<b>Push-Pull Configuration SO and SK Outputs (<math>I_{OH}</math>)</b>				
	$V_{CC} = 7.5\text{V}$ , $V_{OH} = 3.75\text{V}$	-0.85		mA
	$V_{CC} = 5.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-1.1		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 1.0\text{V}$	-1.2		mA
<b>LED Configuration, <math>L_0</math>-<math>L_7</math> Outputs, Low Current Driver Option (<math>I_{OH}</math>)</b>				
	$V_{CC} = 7.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-1.4	-27	mA
	$V_{CC} = 6.0\text{V}$ , $V_{OH} = 2.0\text{V}$	-1.4	-17	mA
	$V_{CC} = 5.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-0.7	-15	mA
<b>LED Configuration, <math>L_0</math>-<math>L_7</math> Outputs, High Current Driver Option (<math>I_{OH}</math>)</b>				
	$V_{CC} = 7.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-2.7	-54	mA
	$V_{CC} = 6.0\text{V}$ , $V_{OH} = 2.0\text{V}$	-2.7	-34	mA
	$V_{CC} = 5.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-1.4	-30	mA
<b>TRI-STATE<sup>®</sup> Configuration, <math>L_0</math>-<math>L_7</math> Outputs, Low Current Driver Option (<math>I_{OH}</math>)</b>				
	$V_{CC} = 7.5\text{V}$ , $V_{OH} = 4.0\text{V}$	-0.7		mA
	$V_{CC} = 5.5\text{V}$ , $V_{OH} = 2.7\text{V}$	-0.6		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 1.5\text{V}$	-0.9		mA
<b>TRI-STATE<sup>®</sup> Configuration, <math>L_0</math>-<math>L_7</math> Outputs, High Current Driver Option (<math>I_{OH}</math>)</b>				
	$V_{CC} = 7.5\text{V}$ , $V_{OH} = 4.0\text{V}$	-1.4		mA
	$V_{CC} = 5.5\text{V}$ , $V_{OH} = 2.7\text{V}$	-1.2		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 1.5\text{V}$	-1.8		mA
Input Load Source Current	$V_{CC} = 5.0\text{V}$ , $V_{IL} = 0\text{V}$	-10	-200	$\mu\text{A}$
CKO Output RAM Power Supply Option Power Requirement	$V_R = 3.3\text{V}$		4.0	mA
TRI-STATE <sup>®</sup> Output Leakage Current		-5	+5	$\mu\text{A}$
<b>Total Sink Current Allowed</b>				
All Outputs Combined D, G Ports			120	mA
$L_7$ - $L_4$			4	mA
$L_3$ - $L_0$			4	mA
All Other Pins			1.5	mA
<b>Total Source Current Allowed</b>				
All I/O Combined			120	mA
$L_7$ - $L_4$			60	mA
$L_3$ - $L_0$			60	mA
Each L Pin			30	mA
All Other Pins			1.5	mA

## AC Electrical Characteristics

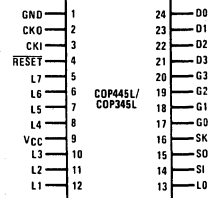
COP444L/445L:  $0^{\circ}\text{C} \leq T_A \leq 70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 9.5\text{V}$  unless otherwise noted.

COP344L/345L:  $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 7.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Instruction Cycle Time — $t_C$		15	40	$\mu\text{s}$
CKI				
Input Frequency — $f_I$	$\div 32$ mode	0.8	2.1	MHz
	$\div 16$ mode	0.4	1.0	MHz
	$\div 8$ mode	0.2	0.5	MHz
	$\div 4$ mode	0.1	0.26	MHz
Duty Cycle		30	60	%
Rise Time	$f_I = 2\text{MHz}$		120	ns
Fall Time			80	ns
CKI Using RC ( $\div 4$ )	$R = 56\text{k}\Omega \pm 5\%$ $C = 100\text{pF} \pm 10\%$			
Instruction Cycle Time		15	28	$\mu\text{s}$
CKO as SYNC Input				
$t_{\text{SYNC}}$		400		ns
INPUTS:				
$\text{IN}_3 - \text{IN}_0, \text{G}_3 - \text{G}_0, \text{L}_7 - \text{L}_0$				
$t_{\text{SETUP}}$			8.0	$\mu\text{s}$
$t_{\text{HOLD}}$			1.3	$\mu\text{s}$
SI				
$t_{\text{SETUP}}$			2.0	$\mu\text{s}$
$t_{\text{HOLD}}$			1.0	$\mu\text{s}$
OUTPUT PROPAGATION DELAY	Test condition: $C_L = 50\text{pF}$ , $R_L = 20\text{k}\Omega$ , $V_{\text{OUT}} = 1.5\text{V}$			
SO, SK Outputs				
$t_{\text{pd1}}$ , $t_{\text{pd0}}$			4.0	$\mu\text{s}$
All Other Outputs				
$t_{\text{pd1}}$ , $t_{\text{pd0}}$			5.6	$\mu\text{s}$



Order Number COP444L/N, COP344L/N  
NS Package N28A



Order Number COP445L/N, COP345L/N  
NS Package N24A

Figure 2. Connection Diagrams

Pin	Description	Pin	Description
L7-L0	8 bidirectional I/O ports with TRI-STATE®	SK	Logic-controlled clock (or general purpose output)
G3-G0	4 bidirectional I/O ports	CKI	System oscillator input
D3-D0	4 general purpose outputs	CKO	System oscillator output (or general purpose input, RAM power supply, or SYNC input)
IN3-IN0	4 general purpose inputs (COP444L only)	RESET	System reset input
SI	Serial input (or counter input)	Vcc	Power supply
SO	Serial output (or general purpose output)	GND	Ground

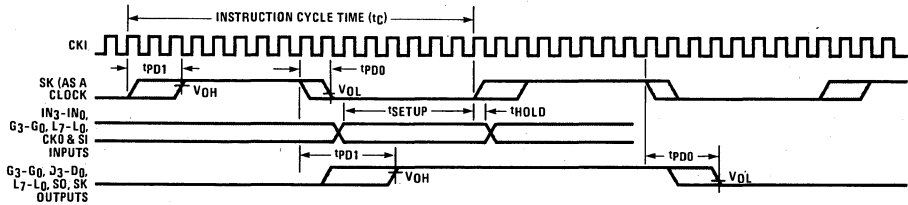


Figure 3. Input/Output Timing Diagrams (Crystal Divide-by-16 Mode)

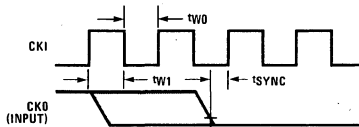


Figure 3a. Synchronization Timing

## Functional Description

A block diagram of the COP444L is given in Figure 1. Data paths are illustrated in simplified form to depict how the various logic elements communicate with each other in implementing the instruction set of the device. Positive logic is used. When a bit is set, it is a logic "1" (greater than 2 volts). When a bit is reset, it is a logic "0" (less than 0.8 volts).

All functional references to the COP444L/COP445L also apply to the COP344L/COP345L.

### Program Memory

Program Memory consists of a 2048 byte ROM. As can be seen by an examination of the COP444L/445L instruction set, these words may be program instructions, program data or ROM addressing data. Because of the special characteristics associated with the JP, JSRP, JID, and LQID instructions, ROM must often be thought of as being organized into 32 pages of 64 words each.

ROM addressing is accomplished by a 11-bit PC register. Its binary value selects one of the 2048 8-bit words contained in ROM. A new address is loaded into the PC register during each instruction cycle. Unless the instruction is a transfer of control instruction, the PC register is loaded with the next sequential 11-bit binary count value. Three levels of subroutine nesting are implemented by the 11-bit subroutine save registers, SA, SB, and SC, providing a last-in, first-out (LIFO) hardware subroutine stack.

ROM instruction words are fetched, decoded and executed by the Instruction Decode, Control and Skip Logic circuitry.

### Data Memory

Data memory consists of a 512-bit RAM, organized as 8 data registers of 16 4-bit digits. RAM addressing is implemented by a 7-bit B register whose upper 3 bits (Br) select 1 of 8 data registers and lower 4 bits (Bd) select 1 of 16 4-bit digits in the selected data register. While the 4-bit contents of the selected RAM digit (M) is usually loaded into or from, or exchanged with, the A register (accumulator), it may also be loaded into or from the Q latches or loaded from the L ports. RAM addressing may also be performed directly by the LDD and XAD instructions based upon the 7-bit contents of the operand field of these instructions. The Bd register also serves as a source register for 4-bit data sent directly to the D outputs.

### Internal Logic

The 4-bit A register (accumulator) is the source and destination register for most I/O, arithmetic, logic and data memory access operations. It can also be used to load the Br and Bd portions of the B register, to load and input 4 bits of the 8-bit Q latch data, to input 4 bits of the 8-bit L I/O port data and to perform data exchanges with the SIO register.

A 4-bit adder performs the arithmetic and logic functions, storing its results in A. It also outputs a carry bit to the 1-bit C register, most often employed to indicate arithmetic overflow. The C register, in conjunction with the XAS instruction and the EN register,

also serves to control the SK output. C can be outputted directly to SK or can enable SK to be a sync clock each instruction cycle time. (See XAS instruction and EN register description, below.)

Four general-purpose inputs,  $IN_3$ - $IN_0$ , are provided.

The D register provides 4 general-purpose outputs and is used as the destination register for the 4-bit contents of Bd. The D outputs can be directly connected to the digits of a multiplexed LED display.

The G register contents are outputs to 4 general-purpose bidirectional I/O ports. G I/O ports can be directly connected to the digits of a multiplexed LED display.

The Q register is an internal, latched, 8-bit register, used to hold data loaded to or from M and A, as well as 8-bit data from ROM. Its contents are output to the L I/O ports when the L drivers are enabled under program control. (See LEI instruction.)

The 8 L drivers, when enabled, output the contents of latched Q data to the L I/O ports. Also, the contents of L may be read directly into A and M. L I/O ports can be directly connected to the segments of a multiplexed LED display (using the LED Direct Drive output configuration option) with Q data being outputted to the Sa-Sg and decimal point segments of the display.

The SIO register functions as a 4-bit serial-in/serial-out shift register or as a binary counter depending on the contents of the EN register. (See EN register description, below.) Its contents can be exchanged with A, allowing it to input or output a continuous serial data stream. SIO may also be used to provide additional parallel I/O by connecting SO to external serial-in/parallel-out shift registers.

The XAS instruction copies C into the SKL latch. In the counter mode, SK is the output of SKL. In the shift register mode, SK outputs SKL ANDed with the clock.

The EN register is an internal 4-bit register loaded under program control by the LEI instruction. The state of each bit of this register selects or deselects the particular feature associated with each bit of the EN register ( $EN_3$ - $EN_0$ ).

1. The least significant bit of the enable register,  $EN_0$ , selects the SIO register as either a 4-bit shift register or a 4-bit binary counter. With  $EN_0$  set, SIO is an asynchronous binary counter, *decrementing* its value by one upon each low-going pulse ("1" to "0") occurring on the SI input. Each pulse must be at least two instruction cycles wide. SK outputs the value of SKL. The SO output is equal to the value of  $EN_3$ . With  $EN_0$  reset, SIO is a serial shift register shifting left each instruction cycle time. The data present at SI goes into the least significant bit of SIO. SO can be enabled to output the most significant bit of SIO each cycle time. (See 4 below.) The SK output becomes a logic-controlled clock.
2. With  $EN_1$  set the  $IN_1$  input is enabled as an interrupt input. Immediately following an interrupt,  $EN_1$  is reset to disable further interrupts.



3. With  $EN_2$  set, the L drivers are enabled to output the data in Q to the L I/O ports. Resetting  $EN_2$  disables the L drivers, placing the L I/O ports in a high-impedance input state.
4.  $EN_3$ , in conjunction with  $EN_0$ , affects the SO output. With  $EN_0$  set (binary counter option selected) SO will output the value loaded into  $EN_3$ . With  $EN_0$  reset (serial shift register option selected), setting

$EN_3$  enables SO as the output of the SIO shift register, outputting serial shifted data each instruction time. Resetting  $EN_3$  with the serial shift register option selected disables SO as the shift register output; data continues to be shifted through SIO and can be exchanged with A via an XAS instruction but SO remains reset to "0". The table below provides a summary of the modes associated with  $EN_3$  and  $EN_0$ .

Enable Register Modes — Bits  $EN_3$  and  $EN_0$ 

$EN_3$	$EN_0$	SIO	SI	SO	SK
0	0	Shift Register	Input to Shift Register	0	If SKL = 1, SK = CLOCK If SKL = 0, SK = 0
1	0	Shift Register	Input to Shift Register	Serial Out	If SKL = 1, SK = CLOCK If SKL = 0, SK = 0
0	1	Binary Counter	Input to Binary Counter	0	If SKL = 1, SK = 1 If SKL = 0, SK = 0
1	1	Binary Counter	Input to Binary Counter	1	If SKL = 1, SK = 1 If SKL = 0, SK = 0

### Interrupt

The following features are associated with the  $IN_1$  interrupt procedure and protocol and must be considered by the programmer when utilizing interrupts.

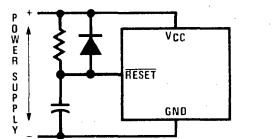
- a. The interrupt, once acknowledged as explained below, pushes the next sequential program counter address ( $PC + 1$ ) onto the stack, pushing in turn the contents of the other subroutine-save registers to the next lower level ( $PC + 1 \rightarrow SA \rightarrow SB \rightarrow SC$ ). Any previous contents of SC are lost. The program counter is set to hex address 0FF (the last word of page 3) and  $EN_1$  is reset.
- b. An interrupt will be acknowledged only after the following conditions are met:
  1.  $EN_1$  has been set.
  2. A low-going pulse ("1" to "0") at least two instruction cycles wide occurs on the  $IN_1$  input.
  3. A currently executing instruction has been completed.
  4. All successive transfer of control instructions and successive LBLs have been completed (e.g., if the main program is executing a JP instruction which transfers program control to another JP instruction, the interrupt will not be acknowledged until the second JP instruction has been executed.
- c. Upon acknowledgement of an interrupt, the skip logic status is saved and later restored upon popping of the stack. For example, if an interrupt occurs during the execution of ASC (Add with Carry, Skip on Carry) instruction which results in carry, the skip logic status is saved and program control is transferred to the interrupt servicing routine at hex address 0FF. At the end of the interrupt routine, a RET instruction is executed to

"pop" the stack and return program control to the instruction following the original ASC. At this time, the skip logic is enabled and skips this instruction because of the previous ASC carry. Subroutines and LQID instructions should not be nested within the interrupt service routine, since their popping the stack will enable any previously saved main program skips, interfering with the orderly execution of the interrupt routine.

- d. The first instruction of the interrupt routine at hex address 0FF must be a NOP.
- e. A LEI instruction can be put immediately before the RET to re-enable interrupts.

### Initialization

The Reset Logic will initialize (clear) the device upon power-up if the power supply rise time is less than 1ms and greater than 1 $\mu$ s. If the power supply rise time is greater than 1ms, the user use provide an external RC network and diode to the RESET pin as shown below. If the RC network is not used, the RESET pin must be pulled up to  $V_{CC}$  either by the internal load or by an external resistor ( $\geq 40k\Omega$ ) to  $V_{CC}$ . The RESET pin is configured as a Schmitt trigger input. Initialization will occur whenever a logic "0" is applied to the RESET input, provided it stays low for at least three instruction cycle times.



Power-Up Clear Circuit

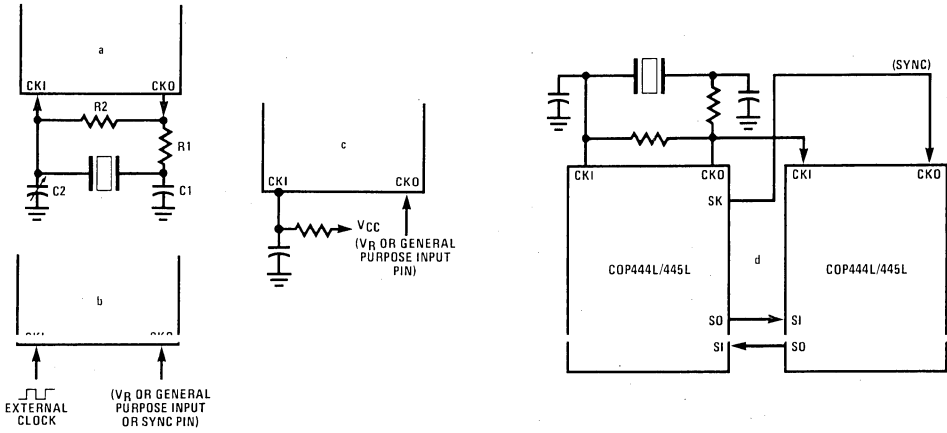
Upon initialization, the PC register is cleared to 0 (ROM address 0) and the A, B, C, D, EN, and G registers are cleared. The SK output is enabled as a SYNC output, providing a pulse each instruction cycle time. *Data Memory (RAM) is not cleared upon initialization.* The first instruction at address 0 must be a CLRA.

**Oscillator**

There are four basic clock oscillator configurations available as shown by Figure 4.

- a. **Crystal Controlled Oscillator.** CKI and CKO are connected to an external crystal. The instruction cycle time equals the crystal frequency divided by 32 (optional by 16 or 8).
- b. **External Oscillator.** CKI is an external clock input signal. The external frequency is divided by 32 (optional by 16 or 8) to give the instruction cycle time. CKO is now available to be used as the RAM power supply ( $V_R$ ), as a general purpose input, or as a SYNC input.

- c. **RC Controlled Oscillator.** CKI is configured as a single pin RC controlled Schmitt trigger oscillator. The instruction cycle equals the oscillation frequency divided by 4. CKO is available as the RAM power supply ( $V_R$ ) or as a general purpose input.
- d. **Externally Synchronized Oscillator.** Intended for use in multi-COP systems, CKO is programmed to function as an input connected to the SK output of another COP chip operating at the same frequency (COP chip with L or C suffix) with CKI connected as shown. In this configuration, the SK output connected to CKO must provide a SYNC (instruction cycle) signal to CKO, thereby allowing synchronous data transfer between the COPs using only the SI and SO serial I/O pins in conjunction with the XAS instruction. Note that on power-up SK is automatically enabled as a SYNC output. (See Functional Description, Initialization, above.)



**Crystal Oscillator**

Crystal Value	Component Values			
	R1 ( $\Omega$ )	R2 ( $\Omega$ )	C1 (pF)	C2 (pF)
455 kHz	4.7k	1M	220	220
2.097 MHz	1k	1M	30	6-36

**RC Controlled Oscillator**

R (k $\Omega$ )	C (pF)	Instruction Cycle Time ( $\mu$ s)
51	100	19 $\pm$ 15%
82	56	19 $\pm$ 13%

**Note:** 200 k $\Omega$   $\geq$  R  $\geq$  25 k $\Omega$   
 360 pF  $\geq$  C  $\geq$  50 pF

Figure 4. COP444L/445L Oscillator

## CKO Pin Options

In a crystal controlled oscillator system, CKO is used as an output to the crystal network. As an option CKO can be a SYNC input as described above. As another option CKO can be a general purpose input, read into bit 2 of A (accumulator) upon execution of an INIL instruction. As another option, CKO can be a RAM power supply pin ( $V_R$ ), allowing its connection to a standby/backup power supply to maintain the integrity of RAM data with minimum power drain when the main supply is inoperative or shut down to conserve power. Using either option is appropriate in applications where the COP444L/445L system timing configuration does not require use of the CKO pin.

## I/O Options

COP444L/445L outputs have the following optional configurations, illustrated in Figure 5:

- a. Standard** — an enhancement mode device to ground in conjunction with a depletion-mode device to  $V_{CC}$ , compatible with LSTTL and CMOS input requirements. Available on SO, SK, and all D and G outputs.
- b. Open-Drain** — an enhancement-mode device to ground only, allowing external pull-up as required by the user's application. Available on SO, SK, and all D and G outputs.
- c. Push-Pull** — An enhancement-mode device to ground in conjunction with a depletion-mode device paralleled by an enhancement-mode device to  $V_{CC}$ . This configuration has been provided to allow for fast rise and fall times when driving capacitive loads. Available on SO and SK outputs only.
- d. Standard L** — same as **a.**, but may be disabled. Available on L outputs only.
- e. Open Drain L** — same as **b.**, but may be disabled. Available on L outputs only.
- f. LED Direct Drive** — an enhancement-mode device to ground and to  $V_{CC}$ , meeting the typical current sourcing requirements of the segments of an LED display. The sourcing device is clamped to limit current flow. These devices may be turned off under program control (See Functional Description, EN Register), placing the outputs in a high-impedance state to provide required LED segment blanking for a multiplexed display. Available on L outputs only.
- g. TRI-STATE® Push-Pull** — an enhancement-mode device to ground and  $V_{CC}$ . These outputs are TRI-STATE outputs, allowing for connection of these outputs to a data bus shared by other bus drivers. Available on L outputs only.

COP444L/COP445L inputs have the following optional configurations:

- h.** An on-chip depletion load device to  $V_{CC}$ .
- i.** A Hi-Z input which must be driven to a "1" or "0" by external components.

The above input and output configurations share common enhancement-mode and depletion-mode devices. Specifically, all configurations use one or more of six devices (numbered 1-6, respectively). Minimum and maximum current ( $I_{OUT}$  and  $V_{OUT}$  curves are given in Figure 6 for each of these devices to allow the designer to effectively use these I/O configurations in designing a system.

The SO, SK outputs can be configured as shown in **a.**, **b.**, or **c.** The D and G outputs can be configured as shown in **a.** or **b.** Note that when inputting data to the G ports, the G outputs should be set to "1." The L outputs can be configured as in **d.**, **e.**, **f.** or **g.**

An important point to remember if using configuration **d.** or **f.** with the L drivers is that even when the L drivers are disabled, the depletion load device will source a small amount of current (see Figure 6, device 2); however, when the L-lines are used as inputs, the disabled depletion device can *not* be relied on to source sufficient current to pull an input to logic "1".

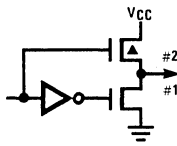
## RAM Keep-Alive Option

Selecting CKO as the RAM power supply ( $V_R$ ) allows the user to shut off the chip power supply ( $V_{CC}$ ) and maintain data in the lower four (Br = 0,1,2,3) registers of RAM. To insure that RAM data integrity is maintained, the following conditions *must* be met:

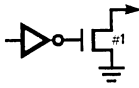
1.  $\overline{RESET}$  must go low before  $V_{CC}$  goes low during power off;  $V_{CC}$  must go high before  $\overline{RESET}$  goes high on power-up.
2.  $V_R$  must be within the operating range of the chip, and equal to  $V_{CC} \pm 1V$  during normal operation.
3.  $V_R$  must be  $\geq 3.3V$  with  $V_{CC}$  off.

## COP445L

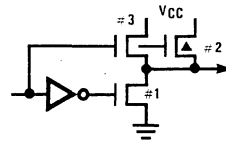
If the COP444L is bonded as a 24-pin device, it becomes the COP445L, illustrated in Figure 2, COP444L/445L Connection Diagrams. Note that the COP445L does not contain the four general purpose IN inputs ( $IN_3 - IN_0$ ). Use of this option precludes, of course, use of the IN options and the interrupt feature, which uses  $IN_1$ . All other options are available for the COP445L.



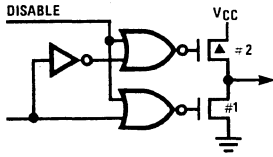
a. Standard Output



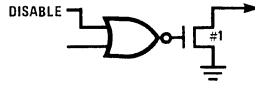
b. Open-Drain Output



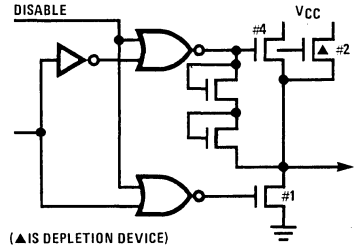
c. Push-Pull Output



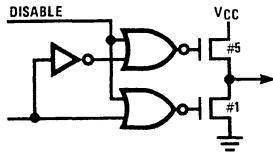
d. Standard L Output



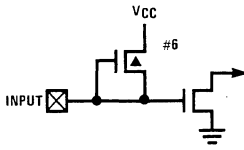
e. Open-Drain L Output



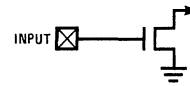
f. LED (L Output)



g. TRI-STATE® Push-Pull (L Output)



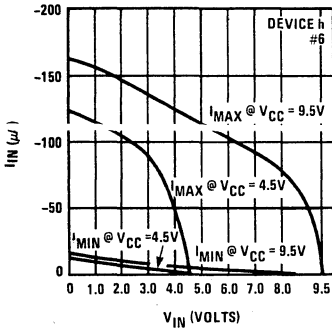
h. Input with Load



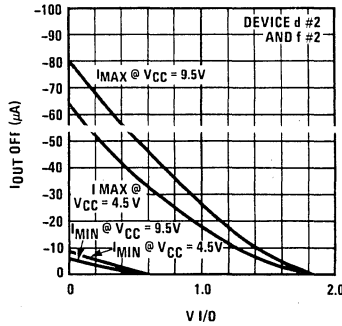
i. Hi-Z Input

Figure 5. Output Configurations

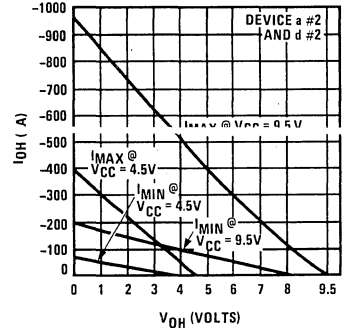
Current for Inputs with Load Device



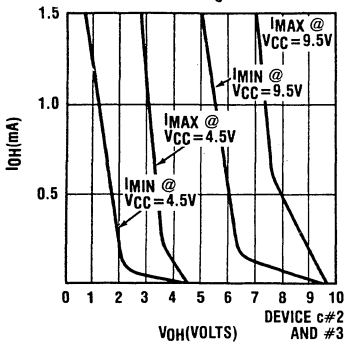
Input Current for L<sub>0</sub> through L<sub>7</sub> when Output Programmed Off by Software



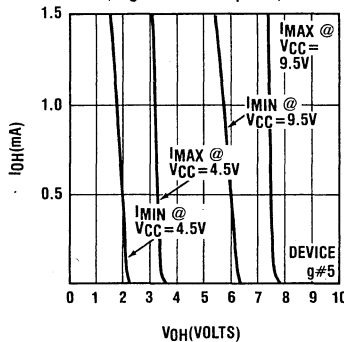
Source Current for Standard Output Configuration



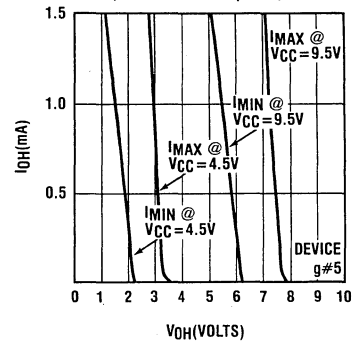
Source Current for SO and SK in Push-Pull Configuration



Source Current for L<sub>0</sub> through L<sub>7</sub> in TRI-STATE® Configuration (High Current Option)



Source Current for L<sub>0</sub> through L<sub>7</sub> in TRI-STATE® Configuration (Low Current Option)



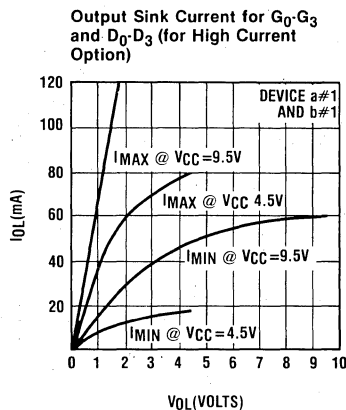
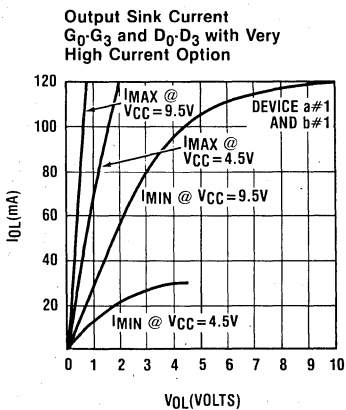
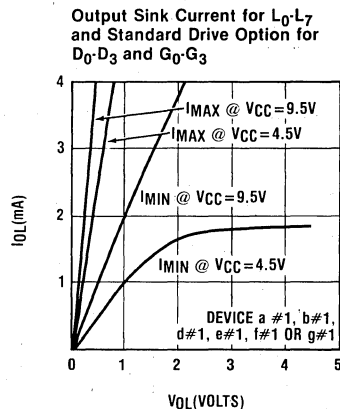
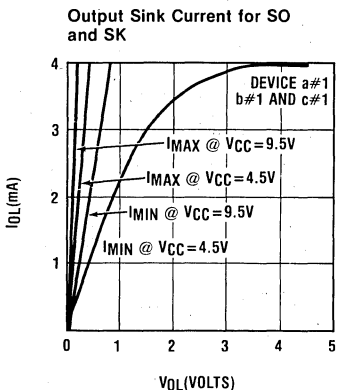
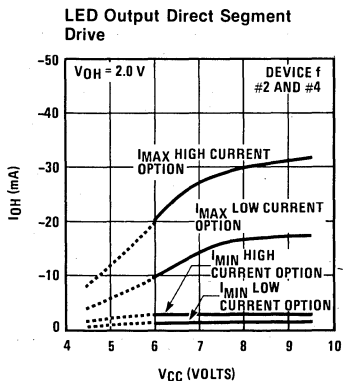
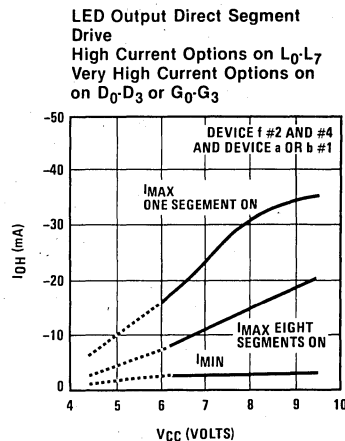
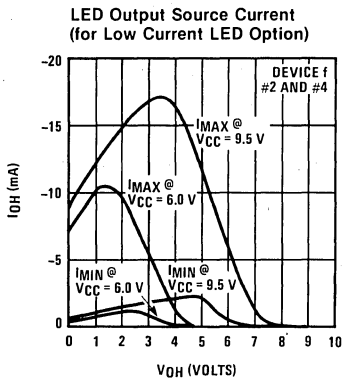
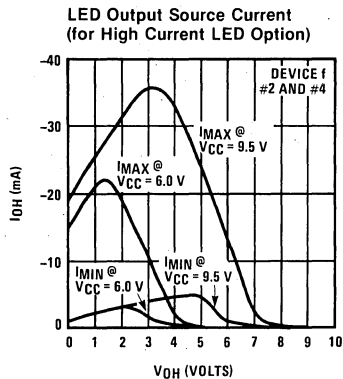


Figure 6a. COP444L/COP445L Input/Output Characteristics

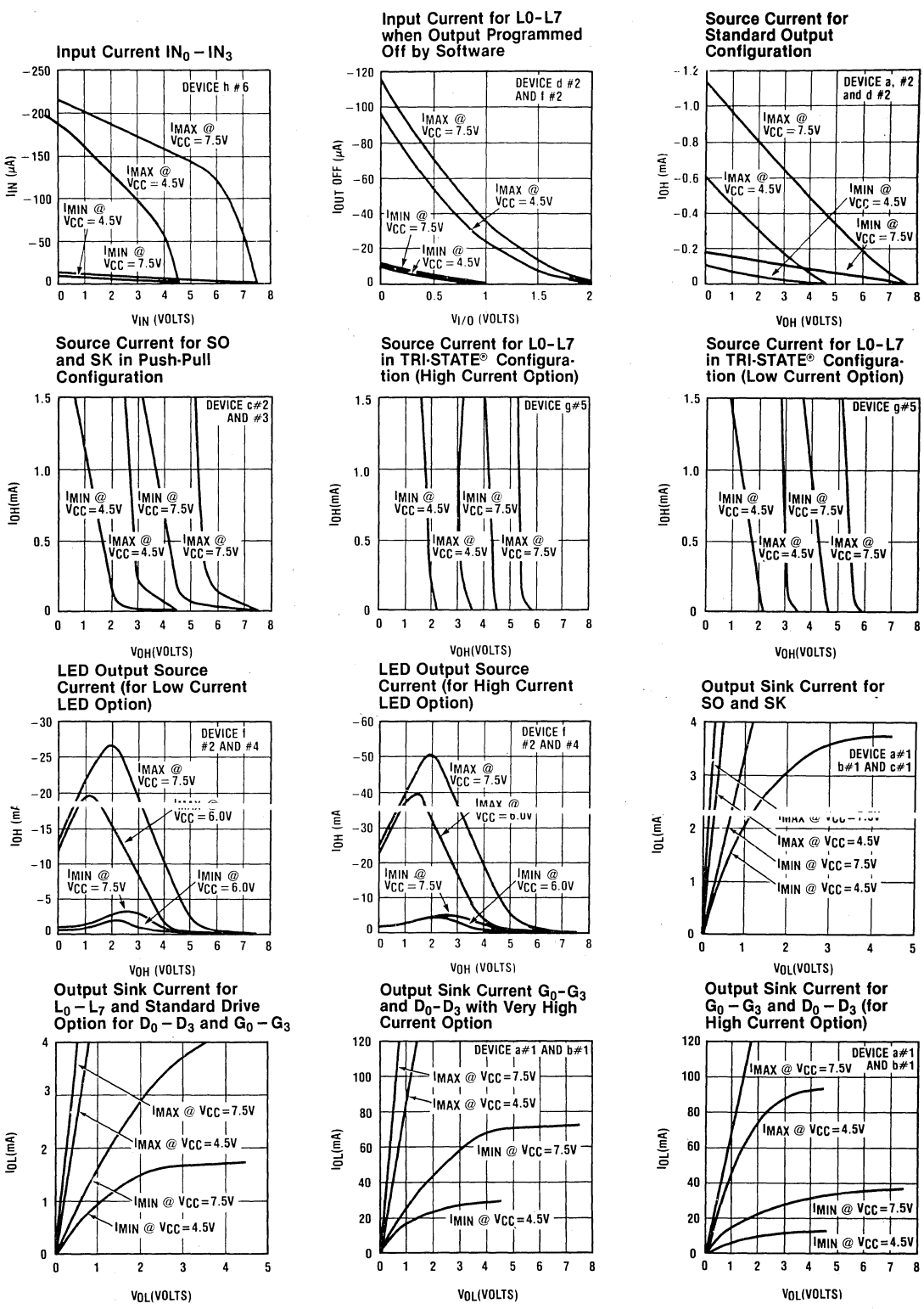


Figure 6b. COP444L/COP445L Input/Output Characteristics

**COP444L/COP445L/COP344L/COP345L Instruction Set**

Table 1 is a symbol table providing internal architecture, instruction operand and operational symbols used in the instruction set table.

Table 2 provides the mnemonic, operand, machine code, data flow, skip conditions, and description associated with each instruction in the COP444L/COP445L instruction set.

Table 1. COP444L/445L/344L/345L Instruction Table Symbols

Symbol	Definition	Symbol	Definition
<b>INTERNAL ARCHITECTURE SYMBOLS</b>		<b>INSTRUCTION OPERAND SYMBOLS</b>	
A	4-bit Accumulator	d	4-bit Operand Field, 0-15 binary (RAM Digit Select)
B	7-bit RAM Address Register	r	3-bit Operand Field, 0-7 binary (RAM Register Select)
Br	Upper 3 bits of B (register address)	a	11-bit Operand Field, 0-2047 binary (ROM Address)
Bd	Lower 4 bits of B (digit address)	y	4-bit Operand Field, 0-15 binary (Immediate Data)
C	1-bit Carry Register	RAM(s)	Contents of RAM location addressed by s
D	4-bit Data Output Port	ROM(t)	Contents of ROM location addressed by t
EN	4-bit Enable Register	<b>OPERATIONAL SYMBOLS</b>	
G	4-bit Register to latch data for G I/O Port	+	Plus
IL	Two 1-bit latches associated with the IN <sub>3</sub> or IN <sub>0</sub> inputs	-	Minus
IN	4-bit Input Port	→	Replaces
L	8-bit TRI-STATE® I/O Port	↔	Is exchanged with
M	4-bit contents of RAM Memory pointed to by B Register	=	Is equal to
PC	11-bit ROM Address Register (program counter)	$\bar{A}$	The one's complement of A
Q	8-bit Register to latch data for L I/O Port	⊕	Exclusive-OR
SA	11-bit Subroutine Save Register A	:	Range of values
SB	11-bit Subroutine Save Register B		
SC	11-bit Subroutine Save Register C		
SIO	4-bit Shift Register and Counter		
SK	Logic-Controlled Clock Output		

Table 2. COP444L/445L Instruction Set

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>ARITHMETIC INSTRUCTIONS</b>						
ASC		30	$\boxed{0011 0000}$	$A + C + \text{RAM}(B) \rightarrow A$ Carry $\rightarrow C$	Carry	Add with Carry, Skip on Carry
ADD		31	$\boxed{0011 0001}$	$A + \text{RAM}(B) \rightarrow A$	None	Add RAM to A
ADT		4A	$\boxed{0100 1010}$	$A + 10_{10} \rightarrow A$	None	Add Ten to A
AISC	y	5-	$\boxed{0101 y}$	$A + y \rightarrow A$	Carry	Add Immediate, Skip on Carry ( $y \neq 0$ )
CASC		10	$\boxed{0001 0000}$	$\bar{A} + \text{RAM}(B) + C \rightarrow A$ Carry $\rightarrow C$	Carry	Complement and Add with Carry, Skip on Carry
CLRA		00	$\boxed{0000 0000}$	$0 \rightarrow A$	None	Clear A
COMP		40	$\boxed{0100 0000}$	$\bar{A} \rightarrow A$	None	Ones complement of A to A
NOP		44	$\boxed{0100 0100}$	None	None	No Operation
RC		32	$\boxed{0011 0010}$	"0" $\rightarrow C$	None	Reset C
SC		22	$\boxed{0010 0010}$	"1" $\rightarrow C$	None	Set C
XOR		02	$\boxed{0000 0010}$	$A \oplus \text{RAM}(B) \rightarrow A$	None	Exclusive-OR RAM with A
<b>TRANSFER OF CONTROL INSTRUCTIONS</b>						
JID		FF	$\boxed{1111 1111}$	ROM ( $\text{PC}_{10:8}, A, M$ ) $\rightarrow \text{PC}_{7:0}$	None	Jump Indirect (Note 3)
JMP	a	6-	$\boxed{0110 0 a_{10:8}}$ $\boxed{a_{7:0}}$	$a \rightarrow \text{PC}$	None	Jump
JP	a	--	$\boxed{1 a_{6:0}}$ (pages 2,3 only)	$a \rightarrow \text{PC}_{6:0}$	None	Jump within Page (Note 4)
		--	$\boxed{11 a_{5:0}}$ (all other pages)	$a \rightarrow \text{PC}_{5:0}$		
JSRP	a	--	$\boxed{10 a_{5:0}}$	$\text{PC} + 1 \rightarrow \text{SA} \rightarrow \text{SB} \rightarrow \text{SC}$ $00010 \rightarrow \text{PC}_{10:6}$ $a \rightarrow \text{PC}_{5:0}$	None	Jump to Subroutine Page (Note 5)
JSR	a	6-	$\boxed{0110 1 a_{10:8}}$ $\boxed{a_{7:0}}$	$\text{PC} + 1 \rightarrow \text{SA} \rightarrow \text{SB} \rightarrow \text{SC}$ $a \rightarrow \text{PC}$	None	Jump to Subroutine
RET		48	$\boxed{0100 1000}$	$\text{SC} \rightarrow \text{SB} \rightarrow \text{SA} \rightarrow \text{PC}$	None	Return from Subroutine
RETSK		49	$\boxed{0100 1001}$	$\text{SC} \rightarrow \text{SB} \rightarrow \text{SA} \rightarrow \text{PC}$	Always Skip on Return	Return from Subroutine then Skip



Table 2. COP444L/445L Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>MEMORY REFERENCE INSTRUCTIONS</b>						
CAMQ		33 3C	$\begin{array}{ c c } \hline 0011 & 0011 \\ \hline 0011 & 1100 \\ \hline \end{array}$	A → Q7:4 RAM(B) → Q3:0	None	Copy A, RAM to Q
CQMA		33 2C	$\begin{array}{ c c } \hline 0011 & 0011 \\ \hline 0010 & 1100 \\ \hline \end{array}$	Q7:4 → RAM(B) Q3:0 → A	None	Copy Q to RAM, A
LD	r	-5	$\begin{array}{ c c } \hline 00 & r & 0101 \\ \hline \end{array}$ (r = 0:3)	RAM(B) → A Br ⊕ r → Br	None	Load RAM into A, Exclusive-OR Br with r
LDD	r,d	23 --	$\begin{array}{ c c c } \hline 0010 & 0011 \\ \hline 0 & r & d \\ \hline \end{array}$	RAM(r,d) → A	None	Load A with RAM pointed to directly by r,d
LQID		BF	$\begin{array}{ c c } \hline 1011 & 1111 \\ \hline \end{array}$	ROM(PC10:8,A,M) → Q SB → SC	None	Load Q Indirect (Note 3)
RMB	0 1 2 3	4C 45 42 43	$\begin{array}{ c c } \hline 0100 & 1100 \\ \hline 0100 & 0101 \\ \hline 0100 & 0010 \\ \hline 0100 & 0011 \\ \hline \end{array}$	0 → RAM(B) <sub>0</sub> 0 → RAM(B) <sub>1</sub> 0 → RAM(B) <sub>2</sub> 0 → RAM(B) <sub>3</sub>	None	Reset RAM Bit
SMB	0 1 2 3	4D 47 46 4B	$\begin{array}{ c c } \hline 0100 & 1101 \\ \hline 0100 & 1101 \\ \hline 0100 & 0110 \\ \hline 0100 & 1011 \\ \hline \end{array}$	1 → RAM(B) <sub>0</sub> 1 → RAM(B) <sub>1</sub> 1 → RAM(B) <sub>2</sub> 1 → RAM(B) <sub>3</sub>	None	Set RAM Bit
STII	y	7-	$\begin{array}{ c c } \hline 0111 & y \\ \hline \end{array}$	y → RAM(B) Bd + 1 → Bd	None	Store Memory Immediate and Increment Bd
X	r	-6	$\begin{array}{ c c } \hline 00 & r & 0110 \\ \hline \end{array}$ (r = 0:3)	RAM(B) ↔ A Br ⊕ r → Br	None	Exchange RAM with A, Exclusive-OR Br with r
XAD	r,d	23 --	$\begin{array}{ c c c } \hline 0010 & 0011 \\ \hline 1 & r & d \\ \hline \end{array}$	RAM(r,d) ↔ A	None	Exchange A with RAM pointed to directly by r,d
XDS	r	-7	$\begin{array}{ c c } \hline 00 & r & 0111 \\ \hline \end{array}$ (r = 0:3)	RAM(B) ↔ A Bd - 1 → Bd Br ⊕ r → Br	Bd decrements past 0	Exchange RAM with A and Decrement Bd, Exclusive-OR Br with r
XIS	r	-4	$\begin{array}{ c c } \hline 00 & r & 0100 \\ \hline \end{array}$ (r = 0:3)	RAM(B) ↔ A Bd + 1 → Bd Br ⊕ r → Br	Bd increments past 15	Exchange RAM with A and Increment Bd, Exclusive-OR Br with r
<b>REGISTER REFERENCE INSTRUCTIONS</b>						
CAB		50	$\begin{array}{ c c } \hline 0101 & 0000 \\ \hline \end{array}$	A → Bd	None	Copy A to Bd
CBA		4E	$\begin{array}{ c c } \hline 0100 & 1110 \\ \hline \end{array}$	Bd → A	None	Copy Bd to A
LBI	r,d	--	$\begin{array}{ c c } \hline 00 & r & (d-1) \\ \hline \end{array}$ (r = 0:3; d = 0, 9:15) or $\begin{array}{ c c } \hline 0011 & 0011 \\ \hline 1 & r & d \\ \hline \end{array}$ (any r, any d)	r,d → B	Skip until not a LBI	Load B Immediate with r,d (Note 6)
LEI	y	33 6-	$\begin{array}{ c c } \hline 0011 & 0001 \\ \hline 0110 & y \\ \hline \end{array}$	y → EN	None	Load EN Immediate (Note 7)
XABR		12	$\begin{array}{ c c } \hline 0001 & 0010 \\ \hline \end{array}$	A ↔ Br (0 → A <sub>3</sub> )	None	Exchange A with Br

Table 2. COP444L/445L Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
TEST INSTRUCTIONS						
SKC		20	<u>0010</u> <u>0000</u>		C = "1"	Skip if C is True
SKE		21	<u>0010</u> <u>0001</u>		A = RAM(B)	Skip if A Equals RAM
SKGZ		33	<u>0011</u> <u>0011</u>		G <sub>3:0</sub> = 0	Skip if G is Zero (all 4 bits)
SKGBZ		33	<u>0011</u> <u>0011</u>	1st byte		Skip if G Bit is Zero
	0	01	<u>0000</u> <u>0001</u>	2nd byte	G <sub>0</sub> = 0	
	1	11	<u>0001</u> <u>0001</u>		G <sub>1</sub> = 0	
	2	03	<u>0000</u> <u>0011</u>		G <sub>2</sub> = 0	
	3	13	<u>0001</u> <u>0011</u>		G <sub>3</sub> = 0	
SKMBZ	0	01	<u>0000</u> <u>0001</u>		RAM(B) <sub>0</sub> = 0	Skip if RAM Bit is Zero
	1	11	<u>0001</u> <u>0001</u>		RAM(B) <sub>1</sub> = 0	
	2	03	<u>0000</u> <u>0011</u>		RAM(B) <sub>2</sub> = 0	
	3	13	<u>0001</u> <u>0011</u>		RAM(B) <sub>3</sub> = 0	
SKT		41	<u>0100</u> <u>0001</u>		A time-base counter carry has occurred since last test	Skip on Timer (Note 3)

INPUT/OUTPUT INSTRUCTIONS						
ING		33	<u>0011</u> <u>0011</u>	G → A	None	Input G Ports to A
		2A	<u>0010</u> <u>1010</u>			
ININ		33	<u>0011</u> <u>0011</u>	IN → A	None	Input IN Inputs to A (Note 2)
		28	<u>0010</u> <u>1000</u>			
INIL		33	<u>0011</u> <u>0011</u>	IL <sub>3</sub> , CKO, "0", IL <sub>0</sub> → A	None	Input IL Latches to A (Note 3)
		29	<u>0010</u> <u>1001</u>			
INL		33	<u>0011</u> <u>0011</u>	L <sub>7:4</sub> → RAM(B)	None	Input L Ports to RAM A
		2E	<u>0010</u> <u>1110</u>	L <sub>3:0</sub> → A		
OBD		33	<u>0011</u> <u>0011</u>	Bd → D	None	Output Bd to D Outputs
		3E	<u>0011</u> <u>1110</u>			
OGI	y	33	<u>0011</u> <u>0011</u>	y → G	None	Output to G Ports Immediate
		5-	<u>0101</u> y			
OMG		33	<u>0011</u> <u>0011</u>	RAM(B) → G	None	Output RAM to G Ports
		3A	<u>0011</u> <u>1010</u>			
XAS		4F	<u>0100</u> <u>1111</u>	A ↔ SIO, C → SKL	None	Exchange A with SIO (Note 3)

**Note 1:** All subscripts for alphabetical symbols indicate bit numbers unless explicitly defined (e.g., Br and Bd are explicitly defined). Bits are numbered 0 to N where 0 signifies the least significant bit (low-order, right-most bit). For example, A<sub>3</sub> indicates the most significant (left-most) bit of the 4-bit A register.

**Note 2:** The ININ instruction is not available on the 24-pin COP445L or COP345L since these devices do not contain the IN inputs.

**Note 3:** For additional information on the operation of the XAS, JID, LQID, INIL, and SKT Instructions, see below.

**Note 4:** The JP instruction allows a jump, while in subroutine pages 2 or 3, to any ROM location within the two-page boundary of pages 2 or 3. The JP instruction, otherwise, permits a jump to a ROM location within the current 64-word page. JP may not jump to the last word of a page.

**Note 5:** A JSRP transfers program control to subroutine page 2 (0010 is loaded into the upper 4 bits of P). A JSRP may not be used when in pages 2 or 3. JSRP may not jump to the last word in page 2.

**Note 6:** LBI is a single-byte instruction if d = 0, 9, 10, 11, 12, 13, 14, or 15. The machine code for the lower 4 bits equals the binary value of the "d" data minus 1, e.g., to load the lower four bits of B (Bd) with the value 9 (1001<sub>2</sub>), the lower 4 bits of the LBI instruction equal 8 (1000<sub>2</sub>). To load 0, the lower 4 bits of the LBI instruction should equal 15 (1111<sub>2</sub>).

**Note 7:** Machine code for operand field y for LEI instruction should equal the binary value to be latched into EN, where a "1" or "0" in each bit of EN corresponds with the selection or deselection of a particular function associated with each bit. (See Functional Description, EN Register.)

The following information is provided to assist the user in understanding the operation of several unique instructions and to provide notes useful to programmers in writing COP444L/445L programs.

### XAS Instruction

XAS (Exchange A with SIO) exchanges the 4-bit contents of the accumulator with the 4-bit contents of the SIO register. The contents of SIO will contain serial-in/serial-out shift register or binary counter data, depending on the value of the EN register. An XAS instruction will also affect the SK output. (See Functional Description, EN Register, above.) If SIO is selected as a shift register, an XAS instruction must be performed once every 4 instruction cycles to effect a continuous data stream.

### JID Instruction

JID (Jump Indirect) is an indirect addressing instruction, transferring program control to a new ROM location pointed to indirectly by A and M. It loads the lower 8 bits of the ROM address register PC with the contents of ROM addressed by the 11-bit word, PC<sub>10:8</sub>, A, M. PC<sub>10</sub>, PC<sub>9</sub> and PC<sub>8</sub> are not affected by this instruction.

Note that JID requires 2 instruction cycles to execute.

### INIL Instruction

INIL (Input IL Latches to A) inputs 2 latches, IL<sub>3</sub> and IL<sub>0</sub> (see Figure 7) and CKO into A. The IL<sub>3</sub> and IL<sub>0</sub> latches are set if a low-going pulse ("1" to "0") has occurred on the IN<sub>3</sub> and IN<sub>0</sub> inputs since the last INIL instruction, provided the input pulse stays low for at least two instruction times. Execution of an INIL inputs IL<sub>3</sub> and IL<sub>0</sub> into A3 and A0 respectively, and resets these latches to allow them to respond to subsequent low-going pulses on the IN<sub>3</sub> and IN<sub>0</sub> lines. If CKO is mask programmed as a general purpose input, an INIL will input the state of CKO into A2. If CKO has not been so programmed, a "1" will be placed in A2. A "0" is always placed in A1 upon the execution of an INIL. The general purpose inputs IN<sub>3</sub>-IN<sub>0</sub> are input to A upon execution of an ININ instruction. (See Table 2, ININ instruction.) INIL is useful in recognizing pulses of short duration or pulses which occur too often to be read conveniently by an ININ instruction.

Note: IL latches are not cleared on reset; IL<sub>3</sub> and IL<sub>0</sub> not input on 445L

### LQID Instruction

LQID (Load Q Indirect) loads the 8-bit Q register with the contents of ROM pointed to by the 11-bit word PC<sub>10</sub>, PC<sub>9</sub>, PC<sub>8</sub>, A, M. LQID can be used for table lookup or code conversion such as BCD to seven-segment. The LQID instruction "pushes" the stack (PC + 1 → SA → SB → SC) and replaces the least significant 8 bits of PC as follows: A → PC<sub>7:4</sub>, RAM(B) → PC<sub>3:0</sub>, leaving PC<sub>10</sub>, PC<sub>9</sub> and PC<sub>8</sub> unchanged. The ROM data pointed to by the new address is fetched and loaded into the Q latches. Next, the stack is "popped" (SC → SB → SA → PC), restoring the saved

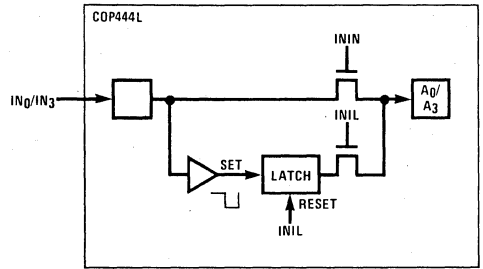


Figure 7. INIL Hardware Implementation

value of PC to continue sequential program execution. Since LQID pushes SB → SC, the previous contents of SC are lost. Also, when LQID pops the stack, the previously pushed contents of SB are left in SC. The net result is that the contents of SB are placed in SC (SB → SC). Note that LQID takes two instruction cycle times to execute.

### SKT Instruction

The SKT (Skip On Timer) instruction tests the state of an internal 10-bit time-base counter. This counter divides the instruction cycle clock frequency by 1024 and provides a latched indication of counter overflow. The SKT instruction tests this latch, executing the next program instruction if the latch is not set. If the latch has been set since the previous test, the next program instruction is skipped and the latch is reset. The features associated with this instruction, therefore, allow the COP444L/445L to generate its own time-base for real-time processing rather than relying on an external input signal.

For example, using a 2.097 MHz crystal as the time-base to the clock generator, the instruction cycle clock frequency will be 65kHz (crystal frequency ÷ 32) and the binary counter output pulse frequency will be 64Hz. For time-of-day or similar real-time processing, the SKT instruction can call a routine which increments a "seconds" counter every 64 ticks.

### Instruction Set Notes

- The first word of a COP444L/445L program (ROM address 0) must be a CLRA (Clear A) instruction.
- Although skipped instructions are not executed, one instruction cycle time is devoted to skipping each byte of the skipped instruction. Thus all program paths except JID and LQID take the same number of cycle times whether instructions are skipped or executed. JID and LQID instructions take 2 cycles if executed and 1 cycle if skipped.
- The ROM is organized into 32 pages of 64 words each. The Program Counter is an 11-bit binary counter, and will count through page boundaries. If a JP, JSRP, JID or LQID instruction is located in the last word of a page, the instruction operates as if it were in the next page. For example: a JP located in the last word of a page will jump to a location in the next page. Also, a LQID or JID located in the last word of page 3, 7, 11, 15, 19, 23, or 27 will access data in the next group of four pages.

## Option List

The COP444L/445L mask-programmable options are assigned numbers which correspond with the COP444L pins.

The following is a list of COP444L options. When specifying a COP445L chip, Options 9, 10, 19, and 20 must all be set to zero. The options are programmed at the same time as the ROM pattern to provide the user with the hardware flexibility to interface to various I/O components using little or no external circuitry.

- Option 1 = 0: Ground Pin — no options available
- Option 2: CKO Output  
 = 0: clock generator output to crystal/resonator (0 not allowable value if option 3 = 3)  
 = 1: pin is RAM power supply ( $V_R$ ) input  
 = 2: general purpose input, load device to  $V_{CC}$   
 = 3: general purpose input, Hi-Z  
 = 4: multi-COP SYNC input (CKI + 32, CKI + 16)  
 = 5: multi-COP SYNC input (CKI + 8)
- Option 3: CKI Input  
 = 0: oscillator input divided by 32 (2MHz max.)  
 = 1: oscillator input divided by 16 (1MHz max.)  
 = 2: oscillator input divided by 8 (500kHz max.)  
 = 3: single-pin RC controlled oscillator divided by 4  
 = 4: oscillator input divided by 4 (Schmitt)
- Option 4:  $\overline{\text{RESET}}$  Input  
 = 0: load device to  $V_{CC}$   
 = 1: Hi-Z input
- Option 5:  $L_7$  Driver  
 = 0: Standard output  
 = 1: Open-drain output  
 = 2: High current LED direct segment drive output  
 = 3: High current TRI-STATE<sup>®</sup> push-pull output  
 = 4: Low-current LED direct segment drive output  
 = 5: Low-current TRI-STATE<sup>®</sup> push-pull output
- Option 6:  $L_6$  Driver  
 same as Option 5
- Option 7:  $L_5$  Driver  
 same as Option 5
- Option 8:  $L_4$  Driver  
 same as Option 5
- Option 9:  $IN_1$  Input  
 = 0: load device to  $V_{CC}$   
 = 1: Hi-Z input
- Option 10:  $IN_2$  Input  
 same as Option 9
- Option 11:  $V_{CC}$  pin  
 = 0: 4.5V to 6.3V operation  
 = 1: 4.5V to 9.5V operation
- Option 12:  $L_3$  Driver  
 same as Option 5
- Option 13:  $L_2$  Driver  
 same as Option 5
- Option 14:  $L_1$  Driver  
 same as Option 5
- Option 15:  $L_0$  Driver  
 same as Option 5
- Option 16: SI Input  
 same as Option 9
- Option 17: SO Driver  
 = 0: standard output  
 = 1: open-drain output  
 = 2: push-pull output
- Option 18: SK Driver  
 same as Option 17
- Option 19:  $IN_0$  Input  
 same as Option 9
- Option 20:  $IN_3$  Input  
 same as Option 9
- Option 21:  $G_0$  I/O Port  
 = 0: very-high current standard output  
 = 1: very-high current open-drain output  
 = 2: high current standard output  
 = 3: high current open-drain output  
 = 4: standard LSTTL output (fanout = 1)  
 = 5: open-drain LSTTL output (fanout = 1)
- Option 22:  $G_1$  I/O Port  
 same as Option 21
- Option 23:  $G_2$  I/O Port  
 same as Option 21
- Option 24:  $G_3$  I/O Port  
 same as Option 21
- Option 25:  $D_3$  Output  
 same as Option 21
- Option 26:  $D_2$  Output  
 same as Option 21
- Option 27:  $D_1$  Output  
 same as Option 21
- Option 28:  $D_0$  Output  
 same as Option 21
- Option 29: L Input Levels  
 = 0: standard TTL input levels ("0" = 0.8V, "1" = 2.0V)  
 = 1: higher voltage input levels ("0" = 1.2V, "1" = 3.6V)
- Option 30: IN Input Levels  
 same as Option 29
- Option 31: G Input Levels  
 same as Option 29
- Option 32: SI Input Levels  
 same as Option 29
- Option 33: RESET Input  
 = 0: Schmitt trigger input  
 = 1: standard TTL input levels  
 = 2: higher voltage input levels
- Option 34: CKO Input Levels (CKO = input; Option 2 = 2,3)  
 same as Option 29
- Option 35 COP Bonding  
 = 0: COP444L (28-pin device)  
 = 1: COP445L (24-pin device)  
 = 2: both 28- and 24-pin versions

### TEST MODE (Non-Standard Operation)

The SO output has been configured to provide for standard test procedures for the custom-programmed COP444L. With SO forced to logic "1," two test modes are provided, depending upon the value of SI:

- RAM and Internal Logic Test Mode (SI = 1)
- ROM Test Mode (SI = 0)

These special test modes should not be employed by the user; they are intended for manufacturing test only.

### APPLICATION #1: COP444L General Controller

Figure 8 shows an interconnect diagram for a COP444L used as a general controller. Operation of the system is as follows:

- The L<sub>7</sub>-L<sub>0</sub> outputs are configured as LED Direct Drive outputs, allowing direct connection to the segments of the display.
- The D<sub>3</sub>-D<sub>0</sub> outputs drive the digits of the multiplexed display directly and scan the columns of the 4 × 4 keyboard matrix.
- The IN<sub>3</sub>-IN<sub>0</sub> inputs are used to input the 4 rows of the keyboard matrix. Reading the IN lines in conjunction with the current value of the D outputs allows detection, debouncing, and decoding of any one of the 16 keyswitches.
- CKI is configured as a single-pin oscillator input allowing system timing to be controlled by a single-pin RC network. CKO is therefore available for use as a general-purpose input.
- SI is selected as the input to a binary counter input. With SIO used as a binary counter, SO and SK can be used as general purpose outputs.
- The 4 bidirectional G I/O ports (G<sub>3</sub>-G<sub>0</sub>) are available for use as required by the user's application.
- Normal reset operation is selected.

### COP444L Evaluation (See COP Note 4)

The 444L-EVAL is a pre-programmed COP444L, containing several routines which facilitate user familiarization and evaluation of the COP444L operating characteristics. It may be used as an up/down counter or timer, interfacing to any combination of (1) an LED digit or lamps, (2) 4-digit LED Display Controller, (3) a 4-digit VF Display Controller, and/or (4) a 4-digit LCD Display Controller, alternatively, it may be used as a simple music synthesizer.

### Sample Circuits

- By making only the oscillator, power supply and "L7" connections, (Fig. 9) an approximate 1Hz square wave will be produced at output "D1." This output may be observed with an oscilloscope, or connected to additional TTL or CMOS circuitry.
- By making the indicated connections to a small LED digit (NSA1541A, NSA1166, or equiv. — larger digits will be proportionately dimmer), the counter actions may be observed. Place the "up/down" switch in the "up" (open) position and apply a TTL-compatible signal at the "counter input." Placing the "up/down" switch in the "down" (closed) position causes the count to decrement on each high-to-low input transition.
- All 4 digits of the counter may be displayed by connecting a standard display controller (COP470 for VF, COP472 for LCD, MM5450 for LED) as shown in Figure 9.

Any combination of the single LED digit and display controllers may be used simultaneously, and will display the same data.

- The simple counter described above becomes a timer when the 1Hz output is connected to the "counter input." Up or down counting may be used with input frequencies up to 1kHz. Improved timing accuracies may be obtained by substituting the

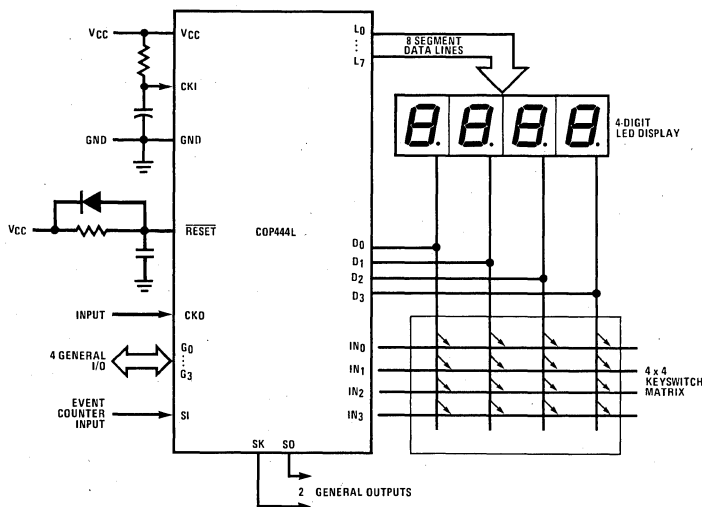


Figure 8. COP444L Keyboard/Display Interface

2.097MHz crystal oscillator circuit of Figure 4a for the RC network shown in Figure 9, or by connecting a more stable external frequency to the "counter input" in place of the 1Hz signal.

5. An "entertaining" use of the 444L-EVAL is a simple music synthesizer (or electronic organ). By attaching a simple switch matrix (or keyboard), a speaker or piezo-ceramic transducer, and grounding "L7", the user can play "music" (Figure 10). Three modes of operation are available: Play a note, play one of four stored tunes, or record a tune for subsequent replay.

a. Play A Note

Twelve keys, representing the 12 notes in one octave, are labeled "C" through "B"; depressing a key causes a square wave of the corresponding frequency to be outputted to the speaker. Depressing "LShift" or "UShift" causes the next note to be shifted to the next lower octave (one-half frequency) or the next upper octave (double frequency), respectively.

b. Play Stored Tune

Depressing "Play" followed by "1/8", "1/4", "1/2", or "1" will cause one of 4 stored tunes to be played.

c. Record Tune

Any combination of notes and rests up to a total of 48 may be stored in RAM for later replay. To store a note, press the appropriate note key, followed by the duration of the note (1/8-note, 1/4-note, 1/2-note, whole (1)-note, followed by "Store;" a rest is stored by selecting the duration and pressing "Store." When the tune is complete, press "Play"

followed by "Store;" the tune will be played for immediate audition. Subsequent depression of "Play" and "Store" will replay the last stored tune.

**Note:** The accuracy of the tones produced is a function of the oscillator accuracy and stability; the crystal oscillator is recommended.

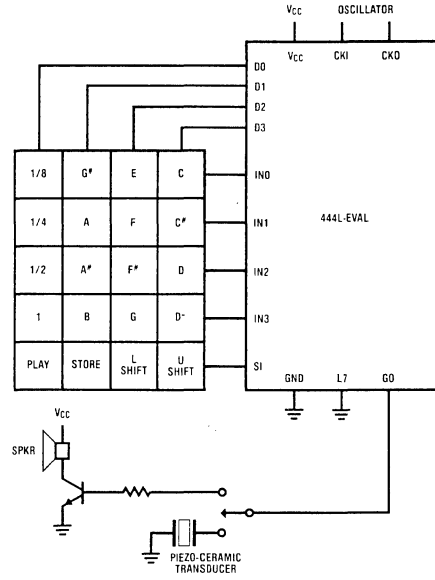


Figure 9. Counter/Timer

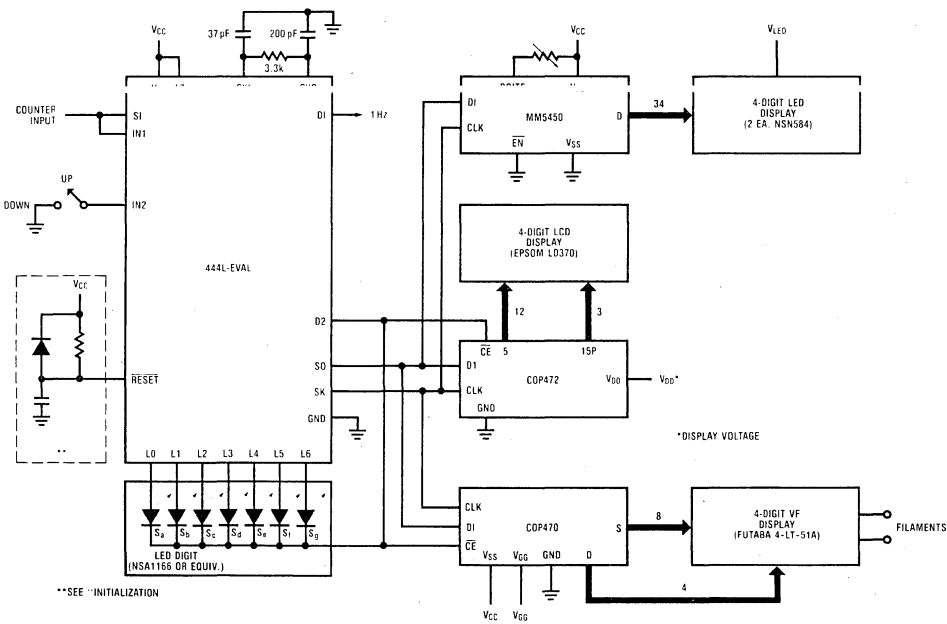


Figure 10. Music Synthesizer

## COP464 and COP484 Single-Chip 3k and 4k Microcontrollers (COP464/COP465, COP364/COP365 and COP484/COP485, COP384/COP385)

### General Description

The COP464, COP465, COP484 and COP485 Single-Chip Microcontrollers are members of the COPST<sup>™</sup> family, fabricated using National's XMOS-II technology. These microcontrollers contain all system timing, internal logic, ROM, RAM and I/O necessary to implement dedicated control functions in a variety of applications. Features include single supply operation, various output configuration options, and an instruction set, internal architecture and I/O scheme designed to facilitate keyboard input, display output and data manipulation. The COP464/465 have 3k of on-chip ROM and 192 digits of RAM, the COP484/485 have 4k of ROM and 256 digits of RAM. The COP464 and COP484 are 28-pin chips. The COP465 and COP485 are 24-pin versions (four inputs removed). The COP364/365 and COP384/385 are functional equivalents of the above devices, but operate with an extended temperature range (-40°C to +85°C). Standard test procedures and reliable high-density fabrication techniques provide the medium-to-large volume customers with a customized microcontroller at a low end-product cost. These microcontrollers are appropriate choices in many demanding control environments, especially those with human interface.

COPS and MICROWIRE are trademarks of National Semiconductor Corp.  
TRI-STATE is a registered trademark of National Semiconductor Corp.

### Features

- Low cost
- Powerful instruction set
- 4k × 8 ROM, 256 × 4 RAM (COP484/485)
- 3k × 8 ROM, 192 × 4 RAM (COP464/465)
- 23 I/O lines (COP464 and COP484)
- True vectored interrupt, plus restart
- Four-level subroutine stack
- 4μs execution time
- Single supply operation (4.5V-6.3V)
- Low current drain (14mA at 25°C)
- Standby current = 2mA at 3.3V (Keep entire RAM alive.)
- Time-base counter for real-time processing
- Internal binary counter/register with MICROWIRE<sup>™</sup>-compatible serial I/O
- General purpose and TRI-STATE<sup>®</sup> outputs
- TTL/CMOS-compatible in and out
- LED drive capability
- Software/hardware compatible with other members of COP400 family
- Extended temperature range devices COP364/365 and COP384/385 (-40°C to 85°C)

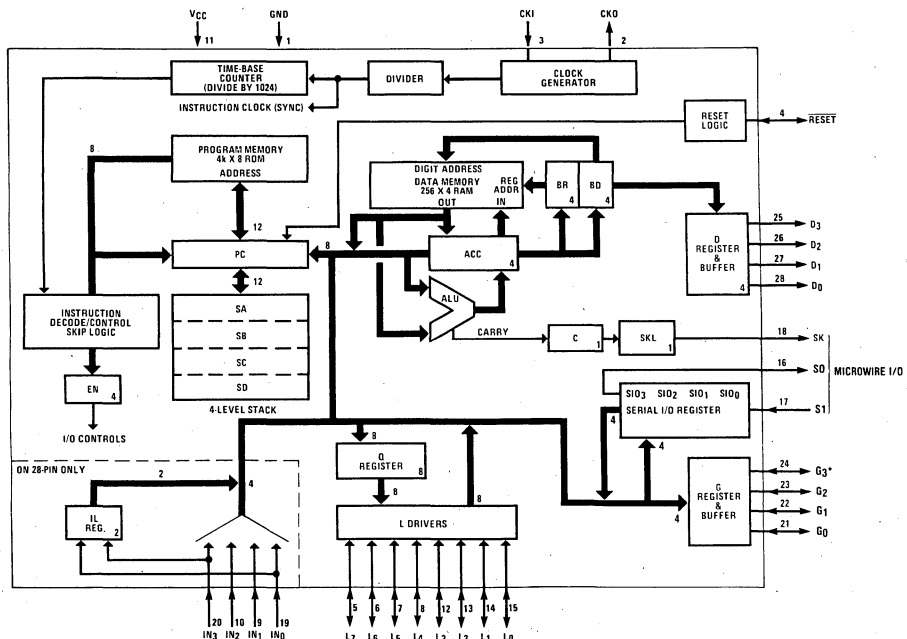


Figure 1. COP484/COP485 Block Diagram

# COP2440/COP2441/COP2442 and COP2340/COP2341/COP2342 Single-Chip Dual CPU Microcontrollers

## General Description

The COP2440, COP2441, COP2442, COP2340, COP2341, and COP2342 Single-Chip Dual CPU Microcontrollers are members of the COPST<sup>™</sup> family, fabricated using N-channel, silicon gate MOS technology. These microcontrollers contain two identical CPUs with all system timing, internal logic, ROM, RAM, and I/O necessary to implement dedicated control functions in a variety of applications. Features include single supply operation, various output configuration options, and an instruction set, internal architecture, and I/O scheme designed to facilitate keyboard input, display output, and data manipulation. The COP2440 is a 40-pin chip and the COP2441 is a 28-pin version of the same circuit (12 I/O lines removed). The COP2442 is a 24-pin version (4 more input lines removed). The COP2340, COP2341, COP2342 are functional equivalents of the above devices respectively, but operate with an extended temperature range (-40°C to +85°C). Standard test procedures and reliable high-density fabrication techniques provide the medium to large volume customers with a customized dual CPU microcontroller at a low end-product cost.

These microcontrollers are appropriate choices in many demanding control environments, especially those with human interface. Further, the high throughput and MICROBUS<sup>™</sup> I/O facilitate numerous machine interface applications. The two CPUs provide the ability to handle two simultaneous but totally independent real time events on one chip.

## Features

- Two independent processors
- Dual CPU simplifies task partitioning—easy to program
- Enhanced, more powerful instruction set
- 2k × 8 ROM, 160 × 4 RAM
- 35 I/O lines (COP2440)
- Zero-crossing detect circuitry with hysteresis
- True multi-vectored interrupt from 4 selectable sources (plus restart)
- Four-level subroutine stack for each processor (in RAM)
- 4μs execution time per processor (non-overlapping)
- Single supply operation (4.5V–6.3V)
- Programmable time-base counter for real-time processing
- Internal binary counter/register with MICROWIRE<sup>™</sup>-compatible serial I/O
- General purpose and TRI-STATE<sup>®</sup> outputs
- TTL/CMOS-compatible in and out
- LED drive capability
- MICROBUS-compatible
- Software/hardware compatible with other members of the COP400 family
- Extended temperature range devices COP2340, COP2341, COP2342 (-40°C to +85°C)
- Compatible single-processor device available (COP440 series)

COPS, MICROBUS, and MICROWIRE are trademarks of National Semiconductor Corp.  
TRI-STATE is a registered trademark of National Semiconductor Corp.

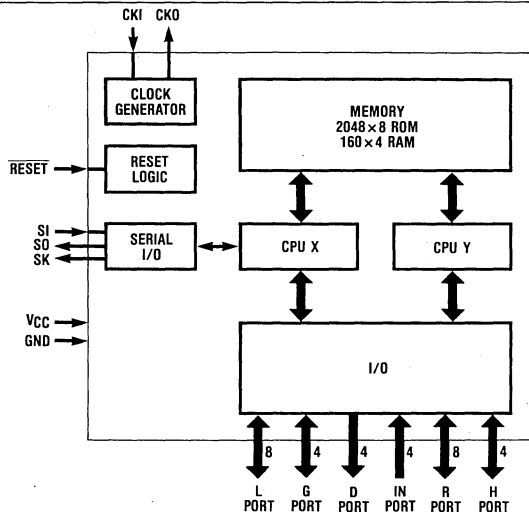


Figure 1. COP2440 Architecture



## COP2440/COP2441/COP2442

### Absolute Maximum Ratings

Voltage at Zero-Crossing Detect Pin Relative to GND	-1.2V to +15V
Voltage at Any Other Pin Relative to GND	-0.5V to +7V
Ambient Operating Temperature	0°C to +70°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C
Power Dissipation	0.75 Watt at 25°C 0.4 Watt at 70°C
Total Source Current	150 mA
Total Sink Current	75 mA

*Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

### DC Electrical Characteristics $0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ , $4.5\text{V} \leq V_{CC} \leq 6.3\text{V}$ unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Operating Voltage ( $V_{CC}$ )	Note 3	4.5	6.3	V
Power Supply Ripple	(peak to peak)		0.4	V
Operating Supply Current	(All inputs and outputs open) $T_A = 0^{\circ}\text{C}$ $T_A = 25^{\circ}\text{C}$ $T_A = 70^{\circ}\text{C}$		41 35 27	mA mA mA
Input Voltage Levels				
CKI Input Levels				
Crystal Input ( $\pm 16, \pm 8$ )				
Logic High ( $V_{IH}$ )	$V_{CC} = \text{Max.}$	2.5		V
Logic High ( $V_{IH}$ )	$V_{CC} = 5\text{V} \pm 5\%$	2.0		V
Logic Low ( $V_{IL}$ )		-0.3	0.4	V
Schmitt Trigger Input ( $\pm 4$ )				
Logic High ( $V_{IH}$ )		$0.7V_{CC}$		V
Logic Low ( $V_{IL}$ )		-0.3	0.6	V
RESET Input Levels	(Schmitt Trigger Input)			
Logic High		$0.7V_{CC}$		V
Logic Low		-0.3	0.6	V
Zero-Crossing Detect Input				
Trip Point	See Figure 9	-0.15	0.15	V
Logic High ( $V_{IH}$ ) Limit			12	V
Logic Low ( $V_{IL}$ ) Limit		-0.8		V
SO Input Level (Test Mode)		2.0	2.5	V
All Other Inputs				
Logic High	$V_{CC} = \text{Max.}$	2.5		V
Logic High	$V_{CC} = 5\text{V} \pm 5\%$	2.0		V
Logic Low		-0.3	0.8	V
Input Levels High Trip Option				
Logic High		3.6		V
Logic Low		-0.3	1.2	V
Input Capacitance			7.0	pF
Hi-Z Input Leakage		-1.0	+1.0	$\mu\text{A}$

**COP2440/COP2441/COP2442**  
**DC Electrical Characteristics** (Cont'd)

Parameter	Conditions	Min.	Max.	Units
<b>Output Voltage Levels</b>				
Standard Output				
TTL Operation				
Logic High ( $V_{OH}$ )	$I_{OH} = -100 \mu A$	2.4		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 1.6 \text{ mA}$		0.4	V
CMOS Operation				
Logic High ( $V_{OH}$ )	$I_{OH} = -10 \mu A$	$V_{CC} - 0.4$		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 10 \mu A$		0.2	V
<b>Output Current Levels</b>				
Standard Output Source Current	$V_{CC} = 4.5V, V_{OH} = 2.4V$	-100	-650	$\mu A$
LED Direct Drive Output	$V_{CC} = 6V$			
Logic High ( $I_{OH}$ )	$V_{OH} = 2V$	-2.5	-17	mA
TRI-STATE® Output Leakage Current		-2.5	+2.5	$\mu A$
CKO Output				
Oscillator Output Option				
Logic High	$V_{OH} = 2V$	-0.2		mA
Logic Low	$V_{OL} = 0.4V$	0.4		mA
$V_R$ RAM Power Supply Option				
Supply current	$V_R = 3.3V$		3.0	mA
CKI Sink Current (RC Option)	$V_{IH} = 3.5V, V_{CC} = 4.5V$	2.0		mA
<b>Input Current Levels</b>				
Zero-Crossing Detect Input				
Resistance	$V_{IH} = 1.0V$	1.5	4.6	k $\Omega$
Input Load Source Current	$V_{IH} = 2.0V, V_{CC} = 4.5V$	14	230	$\mu A$
<b>Total Sink Current Allowed</b>				
All I/O Combined				
Each L, R Port			75	mA
Each D, G, H Port			20	mA
SO, SK			10	mA
			2.5	mA
<b>Total Source Current Allowed</b>				
All I/O Combined				
L Port			150	mA
L7-L4			120	mA
L3-L0			70	mA
Each L Pin			70	mA
All Other Output Pins			23	mA
			1.6	mA

COP2440/COP2441/COP2442, COP2340/COP2341/COP2342

2

**COP2340/COP2341/COP2342****Absolute Maximum Ratings**

Voltage at Zero-Crossing Detect Pin Relative to GND	-1.2V to +15V
Voltage at Any Other Pin Relative to GND	-0.5V to +7V
Ambient Operating Temperature	-40°C to +85°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C
Power Dissipation	0.75 Watt at 25°C 0.25 Watt at 85°C
Total Source Current	150 mA
Total Sink Current	75 mA

*Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

**DC Electrical Characteristics**  $-40^{\circ}\text{C} < T_A < +85^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 5.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Operating Voltage ( $V_{CC}$ )	Note 3	4.5	5.5	V
Power Supply Ripple	(peak to peak)		0.4	V
Operating Supply Current	(All inputs and outputs open) $T_A = -40^{\circ}\text{C}$ $T_A = 25^{\circ}\text{C}$ $T_A = 85^{\circ}\text{C}$		54 35 25	mA mA mA
Input Voltage Levels				
CKI Input Levels				
Crystal Input ( $\pm 8$ )				
Logic High ( $V_{IH}$ )		2.2		V
Logic Low ( $V_{IL}$ )		-0.3	0.3	V
Schmitt Trigger Input ( $\pm 4$ )				
Logic High ( $V_{IH}$ )		$0.7V_{CC}$		V
Logic Low ( $V_{IL}$ )		-0.3	0.4	V
$\overline{\text{RESET}}$ Input Levels	(Schmitt Trigger Input)			
Logic High		$0.7V_{CC}$		V
Logic Low		-0.3	0.4	V
Zero-Crossing Detect Input	See figure 9			
Trip Point		-0.15	0.15	V
Logic High ( $V_{IH}$ ) Limit			12	V
Logic Low ( $V_{IL}$ ) Limit		-0.8		V
SO Input Level (Test Mode)		2.2	2.4	V
All Other Inputs				
Logic High		2.2		V
Logic Low		-0.3	0.6	V
Input Levels High Trip Option				
Logic High		3.6		V
Logic Low		-0.3	1.2	V
Input Capacitance			7.0	pF
Hi-Z Input Leakage		-2.0	+2.0	$\mu\text{A}$

## DC Electrical Characteristics (Cont'd)

Parameter	Conditions	Min.	Max.	Units
Output Voltage Levels				
Standard Output				
TTL Operation				
Logic High ( $V_{OH}$ )	$I_{OH} = -100\mu A$	2.4		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 1.6\text{ mA}$		0.4	V
CMOS Operation				
Logic High ( $V_{OH}$ )	$I_{OH} = -10\mu A$	$V_{CC} - 0.5$		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 10\mu A$		0.2	V
Output Current Levels				
Standard Output Source Current	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 2.4\text{V}$	-100	-800	$\mu A$
LED Direct Drive Output	$V_{CC} = 5\text{V}$ (Note 4)			
Logic High ( $I_{OH}$ )	$V_{OH} = 2\text{V}$	-1.5	-15	mA
TRI-STATE <sup>®</sup> Output Leakage Current		-5.0	+5.0	$\mu A$
CKO Output				
Oscillator Output Option				
Logic High	$V_{OH} = 2\text{V}$	-0.2		mA
Logic Low	$V_{OL} = 0.4\text{V}$	0.4		mA
$V_R$ RAM Power Supply Option				
Supply current	$V_R = 3.3\text{V}$		4.0	mA
CKI Sink Current (RC Option)	$V_{CC} = 4.5\text{V}$ , $V_{IH} = 3.5\text{V}$	2.0		mA
Input Current Levels				
Zero-Crossing Detect Input				
Resistance	$V_{IH} = 1.0\text{V}$	1.4	4.6	k $\Omega$
Input Load Source Current	$V_{IH} = 2.0\text{V}$ , $V_{CC} = 4.5\text{V}$	14	280	$\mu A$
Total Sink Current Allowed				
All I/O Combined			75	mA
Each L, R Port			20	mA
Each D, G, H Port			10	mA
SO, SK			2.5	mA
Total Source Current Allowed				
All I/O Combined			150	mA
L Port			120	mA
L <sub>7</sub> -L <sub>4</sub>			70	mA
L <sub>3</sub> -L <sub>0</sub>			70	mA
Each L Pin			23	mA
All Other Output Pins			1.6	mA

**AC Electrical Characteristics**COP2440/COP2441/COP2442:  $0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 6.3\text{V}$  unless otherwise noted.COP2340/COP2341/COP2442:  $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 5.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Instruction Execution Time — $t_E$	Each Processor (Figure 3)	4.0	10	$\mu\text{s}$
CKI Frequency	+16 mode	1.6	4.0	MHz
	+8 mode	0.8	2.0	MHz
	+4 mode	0.4	1.0	MHz
	$f_i = 4\text{MHz}$	30	60	%
Duty Cycle (Note 1)	$f_i = 4\text{MHz}$ external clock		60	ns
Rise Time	$f_i = 4\text{MHz}$ external clock		40	ns
Fall Time				
CKI Using RC (Figure 11c)	+4 mode			
Frequency	$R = 15\text{k}\Omega \pm 5\%$ , $C = 100\text{pF} \pm 10\%$	0.5	1.0	MHz
Instruction Execution Time — $t_E$		4.0	8.0	$\mu\text{s}$
<b>INPUTS: (Figure 3)</b>				
SI				
$t_{\text{SETUP}}$		0.3		$\mu\text{s}$
$t_{\text{HOLD}}$		300		ns
All Other Inputs				
$t_{\text{SETUP}}$		1.7		$\mu\text{s}$
$t_{\text{HOLD}}$		300		ns
<b>OUTPUT PROPAGATION DELAY</b>				
	Test Condition: $C_L = 50\text{pF}$ , $V_{\text{OUT}} = 1.5\text{V}$			
CKO				
$t_{\text{pd1}}$ , $t_{\text{pd0}}$	Crystal Input		0.17	$\mu\text{s}$
$t_{\text{pd1}}$ , $t_{\text{pd0}}$	Schmitt Trigger Input		0.3	$\mu\text{s}$
SO, SK				
$t_{\text{pd1}}$ , $t_{\text{pd0}}$	$R_L = 2.4\text{k}\Omega$		1.0	$\mu\text{s}$
All Other Outputs	$R_L = 5.0\text{k}\Omega$		1.4	$\mu\text{s}$
<b>MICROBUS™ TIMING</b>				
	$C_L = 100\text{pF}$ , $V_{CC} = 5\text{V} \pm 5\%$ TRI-STATE® outputs			
Read Operation (Figure 6)				
Chip Select Stable Before $\overline{\text{RD}}$ — $t_{\text{CSR}}$		65		ns
Chip Select Hold Time for $\overline{\text{RD}}$ — $t_{\text{RCS}}$		20		ns
$\overline{\text{RD}}$ Pulse Width— $t_{\text{RR}}$		400		ns
Data Delay from $\overline{\text{RD}}$ — $t_{\text{RD}}$			375	ns
$\overline{\text{RD}}$ to Data Floating— $t_{\text{DF}}$			250	ns
Write Operation (Figure 7)				
Chip Select Stable Before $\overline{\text{WR}}$ — $t_{\text{CSW}}$		65		ns
Chip Select Hold Time for $\overline{\text{WR}}$ — $t_{\text{WCS}}$		20		ns
$\overline{\text{WR}}$ Pulse Width— $t_{\text{WW}}$		400		ns
Data Set-Up Time for $\overline{\text{WR}}$ — $t_{\text{DW}}$		320		ns
Data Hold Time for $\overline{\text{WR}}$ — $t_{\text{WD}}$		100		ns
INTR Transition Time from $\overline{\text{WR}}$ — $t_{\text{WI}}$			700	ns

**Note 1:** Duty Cycle =  $t_{\text{WI}} / (t_{\text{WI}} + t_{\text{WO}})$ .**Note 2:** See Figure for additional I/O Characteristics.**Note 3:**  $V_{CC}$  voltage change must be less than 0.5V in a 1ms period to maintain proper operation.**Note 4:** Exercise great care not to exceed maximum device power dissipation limits when direct-driving LEDs (or sourcing similar loads) at high temperature.

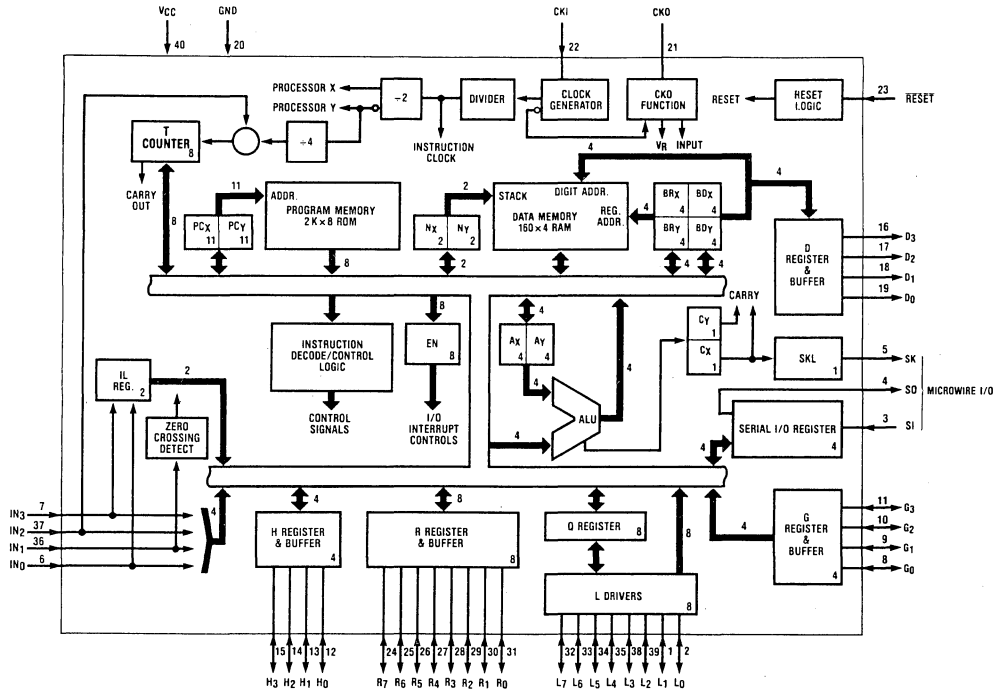


Figure 2. COP2440 Block Diagram

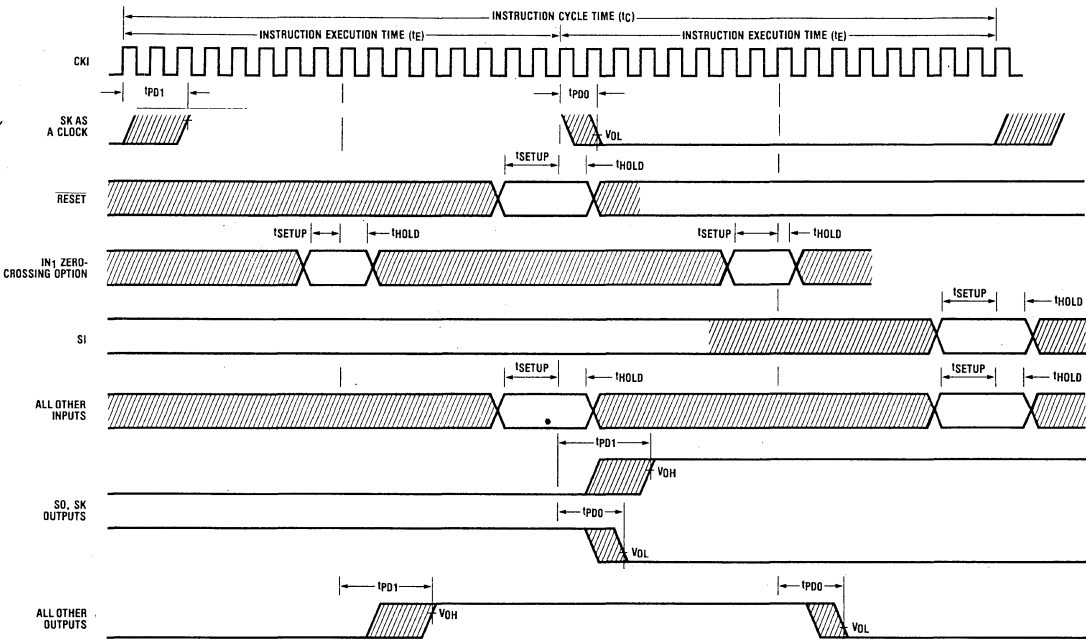
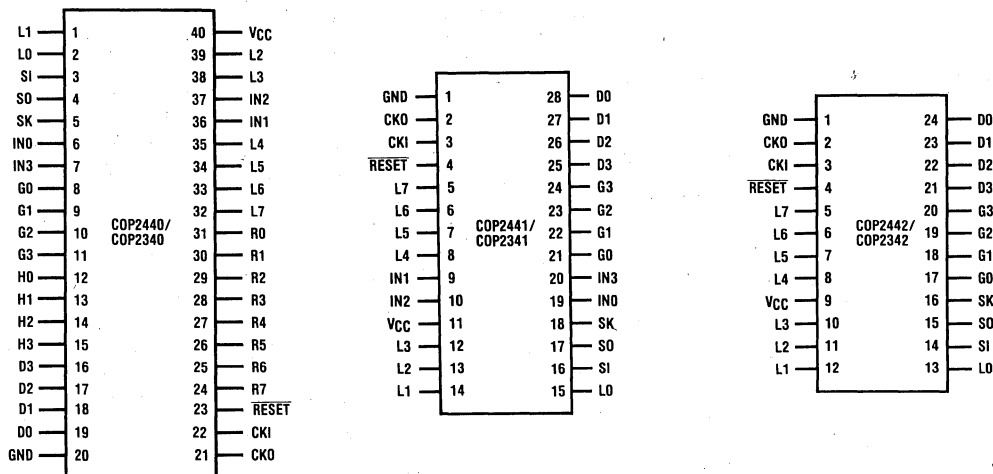


Figure 3. Input/Output Timing Diagrams (Divide by 16 Mode)



Order Number COP2440N, COP2340N  
NS Package N40A

Order Number COP2441N, COP2341N  
NS Package N28A

Order Number COP2442N, COP2342N  
NS Package N24A

Figure 4. Connection Diagrams

Pin	Description	Pin	Description
L <sub>7</sub> -L <sub>0</sub>	8-bit bidirectional I/O port with TRI-STATE®	CKI	System oscillator input
G <sub>3</sub> -G <sub>0</sub>	4-bit bidirectional I/O port	CKO	System oscillator output (or general purpose input or RAM power supply)
D <sub>3</sub> -D <sub>0</sub>	4-bit general purpose output port	RESET	System reset input
IN <sub>3</sub> -IN <sub>0</sub>	4-bit general purpose input port (not available on COP2442/COP2342)	V <sub>CC</sub>	Power supply
SI	Serial input	GND	Ground
SO	Serial output (or general purpose output)	H <sub>3</sub> -H <sub>0</sub>	4-bit bidirectional I/O port (COP2440/COP2340 only)
SK	Logic-controlled clock (or general purpose output)	R <sub>7</sub> -R <sub>0</sub>	8-bit bidirectional I/O port with TRI-STATE® (COP2440/COP2340 only)

## Functional Description

The internal architecture of the COP2440 is shown in Figure 1 and a more detailed block diagram is given in Figure 2. Data paths are illustrated in simplified form to depict how the various logic elements communicate with each other in implementing the instruction set of the device. Positive logic is used. When a bit is set, it is a logic "1" (greater than 2.0 volts). When a bit is reset, it is a logic "0" (less than 0.8 volts).

### Dual Processor

The COP2440 provides an ease of programming and a degree of efficiency not previously available in a single chip microcontroller. The dual CPUs allow easy partitioning of tasks. Simultaneous events can be monitored and handled with ease. Furthermore, each CPU has complete access to all of the ROM and RAM. Both sub-routines and main line codes can be shared and both processors can access the same code simultaneously or at different times so that very efficient programs can be written.

The chip contains two internal processors, X and Y. In order to distinguish between the two processors, start with the **RESET** pin low; the chip is then in the reset mode, with **SK** being a clock output; processor X executes when clock output is high and processor Y executes when the clock output is low. When the **RESET** pin goes high, both X and Y start at location 0 which contains a **CLRA** instruction, then Y jumps to location 401 followed by X to location 1. The processors will then alternately execute 1 byte of code each.

At maximum clock frequency, the *instruction execution time* (single byte instruction) for each processor is 4  $\mu$ s, hence, the *instruction cycle time* for either processor is twice that amount, i.e., 8  $\mu$ s.

Each processor has its own set of status registers: 4-bit A register (accumulator), 8-bit B register (data memory pointer), 1-bit C register (carry), and 2-bit N register (subroutine stack pointer). They function identically except for the following: XAS instruction and SIO register can only be used by X; processor Y treats XAS as a NOP instruction and can only alter bits 2 and 7 of EN register (to enable or disable L and R ports) while leaving the other bits unchanged, hence processor Y does not have the interrupt feature nor access to SO and SK outputs.

### Program Memory

Program Memory consists of a 2,048 byte ROM. As can be seen by an examination of the COP2440 instruction set, these words may be program instructions, constants, or ROM addressing data. Because of the special characteristics associated with the JP, JSRP, JID, LQID, and LID instructions, ROM must often be thought of as being organized into 32 pages of 64 words each.

ROM addressing for each processor is accomplished by its own 11-bit PC register. Its binary value selects one of the 2,048 8-bit words contained in ROM, i.e., each pro-

cessor can address any word in the program memory. A new address is loaded into the PC register during each instruction cycle. Unless the instruction is a transfer of control instruction, the PC register is loaded with the next sequential 11-bit binary count value. Since either of the two processors can address any part of the ROM, they can share any subroutines or codes.

ROM instruction words are fetched, decoded and executed by the Instruction Decode, Control and Skip Logic circuitry.

### Data Memory

Data memory consists of a 640-bit RAM, organized as 10 data registers of 16 4-bit digits. RAM addressing for each processor is implemented by its own 8-bit B register whose upper 4 bits (Br) select 1 of 10 (0-9) data registers and lower 4 bits (Bd) select 1 of 16 4-bit digits in the selected data register. While the 4-bit contents of the selected RAM digit (M) is usually loaded into, or from, or exchanged with the A register (accumulator), it may also be loaded into or from the Q latches, L port, R port, EN register, and T counter (internal time base counter). RAM may also be loaded from 4 bits of a ROM word. RAM addressing may also be performed directly to the lower 8 registers by the LDD and XAD instructions based upon the 7-bit contents of the operand field of these instructions. The Bd register also serves as a source register for 4-bit data sent directly to the D outputs. The upper 2 registers of RAM also serve as subroutine stacks for the two processors. Processor X uses register 8 as its stack, and processor Y uses register 9. Note that it is possible, but not recommended, to alter the contents of the stack by normal data memory access commands.

### Internal Logic

Each processor contains its own 4-bit A register (accumulator) which is the source and destination register for most I/O, arithmetic, logic, and data memory access operations. It can also be used to load the Br and Bd portions of the B register, N register, to load and input 4 bits of the 8-bit Q latch, EN register, or T counter, to input 4 bits of a ROM word, L or R I/O port data, to input 4-bit G, H, or IN ports, and to perform data exchanges with the SIO register. The accumulator is cleared upon reset.

A 4-bit adder performs the arithmetic and logic functions of the COP2440, storing its results in A. It also outputs a carry bit to the 1-bit C register, most often employed to indicate arithmetic overflow. The C register of processor X, in conjunction with the XAS instruction and the EN register, also serves to control the SK output. C can be outputted directly to SK or can enable SK to be a sync clock each instruction cycle time. (See XAS instruction and EN register description, below.)



The 8-bit T counter is a binary up counter which can be loaded to and from M and A. The input to this counter is software selectable from two sources: the first coming from a divide-by-four prescaler (from instruction cycle frequency) thus providing a 10-bit time base counter; the second coming from IN<sub>2</sub> input, changing the T counter into an 8-bit external event counter (see EN register below). In this mode, a low-going pulse ("1" to "0") of at least 1 instruction cycle wide will increment the counter. When the counter overflows, an overflow flag will be set (see SKT instruction below) and an interrupt signal will be sent to processor X. The T counter is cleared on reset.

Four general-purpose inputs, IN<sub>3</sub>-IN<sub>0</sub>, are provided; IN<sub>1</sub>, IN<sub>2</sub> and IN<sub>3</sub> may be selected, by a mask-programmable option, as Read Strobe, Chip Select, and Write Strobe inputs, respectively, for use in MICROBUS™ applications; IN<sub>1</sub>, by another mask-programmable option, can be selected as a true zero-crossing detector with the output triggering an interrupt or being interrogated by an instruction. These two mask-programmable options are mutually exclusive.

The D register provides 4 general-purpose outputs and is used as the destination register for the 4-bit contents of Bd.

The G register contents are outputs to a 4-bit general-purpose bidirectional I/O port. G<sub>0</sub> may be mask-programmed as an output for MICROBUS applications.

The H register contents are outputs to a 4-bit general-purpose bidirectional I/O port.

The Q register is an internal, latched, 8-bit register, used to hold data loaded to or from M and A, as well as 8-bit data from ROM. Its contents are outputted to the L I/O ports when the L drivers are enabled under program control. With the MICROBUS option selected, Q can also be loaded with the 8-bit contents of the L I/O ports upon the occurrence of a write strobe from the host CPU. Note that unlike most other COPS™ controllers, Q is cleared on reset.

The 8 L drivers, when enabled, output the contents of latched Q data to the L I/O ports. Also, the contents of L may be read directly into A and M. As explained above, the MICROBUS option allows L I/O port data to be latched into the Q register. The L I/O port can be directly connected to the segments of a multiplexed LED display (using the LED Direct Drive output configuration option) with Q data being outputted to the Sa-Sg and decimal point segments of the display.

The R register, when enabled, outputs to an 8-bit general-purpose, bidirectional, I/O port.

The SIO register functions as a 4-bit serial-in/serial-out shift register for MICROWIRE™ I/O and COPS peripherals, or as a binary counter for processor X (depending on the contents of the EN register; see EN register description, below). Its contents can be exchanged with A, allowing it to input or output a continuous serial data stream.

The XAS instruction, when executed by processor X, copies the C flag into the SKL latch. In the counter mode, SK is the output of SKL; in the shift register mode, SK outputs SKL ANDed with the instruction cycle clock.

Each processor includes a 2-bit N register which is a stack pointer to the data memory register where the subroutine return address is located. It points to the next location where the address may be stored and increments by 1 after each push of the stack, and decrements by 1 before each pop. The N register can be accessed by exchanging its value with A and is cleared on reset. Processor X uses register 8 and processor Y uses register 9 of the data memory as its stack. Each stack is 4 addresses deep, 12 bits wide, and does not check for overflow or empty conditions. The RAM digit locations where the addresses are stored are shown in Figure 5. The LSBs of the addresses are at digits 0, 4, 8, and 12. The MSBs of digits 2, 6, 10, and 14 contain an interrupt status bit (see Interrupt description, below). The four unused digits (3, 7, 11, and 15) can be used as general data storage. When a subroutine call or interrupt occurs, an 11-bit return address and an interrupt status bit are stored in the stack. The N register is then incremented. When a RET or RETSK instruction is executed, the N register is decremented and then the return address is fetched and loaded into the program counter. The address and interrupt status bits remain in the stack, but will be overwritten when the next subroutine call or interrupt occurs.

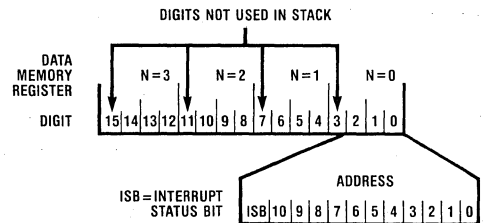


Figure 5. Subroutine Return Address Stack Organization

The EN register is an internal 8-bit register loaded under program control by the LEI instruction (lower 4 bits) or by the CAME instruction. Processor Y can only load bits 2 and 7 of the register. The state of each bit of this register selects or deselects the particular feature associated with each bit of the EN register:

0. The least significant bit of the enable register, EN<sub>0</sub>, selects the SIO register as either a 4-bit shift register or a 4-bit binary counter. With EN<sub>0</sub> set, SIO is an asynchronous binary counter, decrementing its value by one upon each low-going pulse ("1" to "0") occurring on the SI input. Each pulse must be at least two instruction cycles wide. SK outputs the value of SKL. The SO output is equal to the value of EN<sub>3</sub>. With EN<sub>0</sub> reset, SIO is a serial shift register left shifting 1 bit each instruction cycle time. The data present at SI goes into the least significant bit of SIO. SO can be enabled to output the most significant bit of SIO each cycle time. The SK output becomes a logic-controlled clock.

1. With EN<sub>1</sub> set, interrupt is enabled with EN<sub>4</sub> and EN<sub>5</sub> selecting the interrupt source. Immediately following an interrupt, EN<sub>1</sub> is reset to disable further interrupts.
2. With EN<sub>2</sub> set, the L drivers are enabled to output the data in Q to the L I/O port. Resetting EN<sub>2</sub> disables the L drivers, placing the L I/O port in a high-impedance input state. A special feature of the COP2440 and COP2441 is that the MICROBUS™ option will change the function of this bit to disable any writing into G<sub>0</sub> when EN<sub>2</sub> is set.
3. EN<sub>3</sub>, in conjunction with EN<sub>0</sub>, affects the SO output. With EN<sub>0</sub> set (binary counter option selected) SO will output the value loaded into EN<sub>3</sub>. With EN<sub>0</sub> reset (serial shift register option selected), setting EN<sub>3</sub> enables SO as the output of the SIO shift register, outputting serial shifted data each instruction time. Resetting EN<sub>3</sub> with the serial shift register option selected disables SO as the shift register output; data continues to be shifted through SIO and can be exchanged with A via an XAS instruction but SO remains set to "0." Table 1 below provides a summary of the modes associated with EN<sub>3</sub> and EN<sub>0</sub>.

4. EN<sub>4</sub> and EN<sub>5</sub> select the source of the interrupt signal.
5. The possible sources are as follows:

EN <sub>5</sub>	EN <sub>4</sub>	Interrupt Source
0	0	IN <sub>1</sub> (low-going pulse)
0	1	CKO input (if mask-programmed as an input)
1	0	Zero-crossing (or IN <sub>1</sub> level transition)
1	1	T counter overflows

EN<sub>4</sub> determines the interrupt routine location.

6. With EN<sub>6</sub> set, the internal 8-bit T counter will use IN<sub>2</sub> as its input. With EN<sub>6</sub> reset, the input to the T counter is the output of a divide by four prescaler (from instruction cycle frequency), thus providing a 10-bit time-base counter.
7. With EN<sub>7</sub> set, the R outputs are enabled; if EN<sub>7</sub> = 0, the R outputs are disabled.

### Interrupt (Processor X only)

The following features are associated with the interrupt procedure and protocol and must be considered by the programmer when utilizing interrupts.

- a. The interrupt, once acknowledged as explained below, pushes the next sequential program counter address (PC + 1) together with an interrupt status bit, onto the program counter stack residing in data memory. Any previous contents at the bottom of the stack are lost. The program counter is set to hex address 0FF (the last word of page 3) and EN<sub>1</sub> is reset. If EN<sub>4</sub> is reset, the next program address is hex 100; if EN<sub>4</sub> is set, the next program address is hex 300; thus providing a different interrupt location for different interrupt sources.
- b. An interrupt will be acknowledged only after the following conditions are met:
  1. EN<sub>1</sub> has been set.
  2. For an external interrupt input, the signal pulse must be at least one instruction cycle wide.
  3. A currently executing instruction has been completed.
  4. All successive transfer of control instructions and successive LBIs have been completed (e.g., if the main program is executing a JP instruction which transfers program control to another JP instruction, the interrupt will not be acknowledged until the second JP instruction has been executed.

Table 1. Enable Register Modes — Bits EN<sub>3</sub> and EN<sub>0</sub>

EN <sub>3</sub>	EN <sub>0</sub>	SIO	SI	SO	SK
0	0	Shift Register	Input to Shift Register	0	If SKL = 1, SK = Clock If SKL = 0, SK = 0
1	0	Shift Register	Input to Shift Register	Serial Out	If SKL = 1, SK = Clock If SKL = 0, SK = 0
0	1	Binary Counter	Input to Binary Counter	0	If SKL = 1, SK = 1 If SKL = 0, SK = 0
1	1	Binary Counter	Input to Binary Counter	1	If SKL = 1, SK = 1 If SKL = 0, SK = 0

- c. The instruction at hex address 0FF must be a NOP.
- d. A CAME or LEI instruction may be put immediately before the RET instruction to re-enable interrupts.
- e. If the interrupt signal source is being changed, the interrupt must be disabled prior to, or at, the same time with the change to avoid false interrupts. An interrupt may be enabled only if the interrupt source is not changing. A sample code for changing the interrupt source and enabling the interrupt is as follows:

```

CAME      ; disable interrupt & alter interrupt source
SMB 1    ; set interrupt enable bit
CAME      ; enable interrupt
    
```

- f. An interrupt status bit is stored together with the return address in the stack. The status bit is set if an interrupt occurs at a point in the program where the next instruction is to be skipped; upon returning from the interrupt routine, this set status bit will cause the next instruction to be skipped. Subroutine and interrupt nesting inside interrupt routines are allowed. Note that this differs from the COP420/420C/420L/444L series.

**MICROBUS™ Interface**  
(not available in COP2442, COP2342)

The COP2440 series have an option which allows them to be used as peripheral microprocessor devices, inputting and outputting data from and to a host microproces-

sor (μP). IN<sub>1</sub>, IN<sub>2</sub> and IN<sub>3</sub> general purpose inputs become MICROBUS-compatible read-strobe, chip-select, and write-strobe lines, respectively. IN<sub>1</sub> becomes RD — a logic “0” on this input will cause Q latch data to be enabled to the L ports for input to the μP. IN<sub>2</sub> becomes CS — a logic “0” on this line selects the COPST™ processor as the μP peripheral device by enabling the operation of the RD and WR lines and allows for the selection of one of several peripheral components. IN<sub>3</sub> becomes WR — a logic “0” on this line will write bus data from the L ports to the Q latches for input to the COPS processor. G<sub>0</sub> becomes INTR, a “ready” output, reset by a write pulse from the μP on the WR line, providing the “handshaking” capability necessary for asynchronous data transfer between the host CPU and the COPS processor. G<sub>0</sub> output can be separated from other G outputs by the EN<sub>2</sub> bit (see EN description above).

This option has been designed for compatibility with National's MICROBUS™ — a standard interconnect system for 8-bit parallel data transfer between MOS/LSI CPUs and interfacing devices. (See MICROBUS National Publication.) The functional and timing relationships between the COPS processor signal lines affected by this option are as specified for the MICROBUS interface, and are given in the AC electrical characteristics and shown in the timing diagrams (Figures 6 and 7). Connection of the COP2440 to the MICROBUS is shown in Figure 8.

Note: TRI-STATE® outputs must be used on L port.

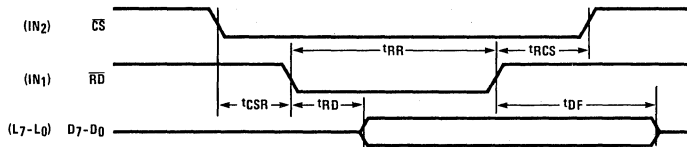


Figure 6. MICROBUS™ Read Operation Timing

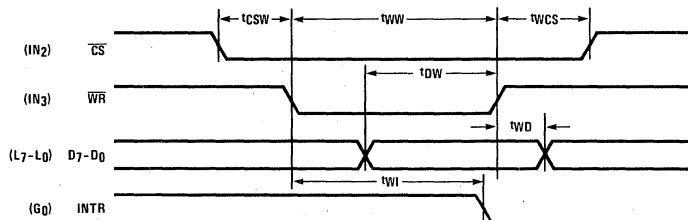


Figure 7. MICROBUS Write Operation Timing

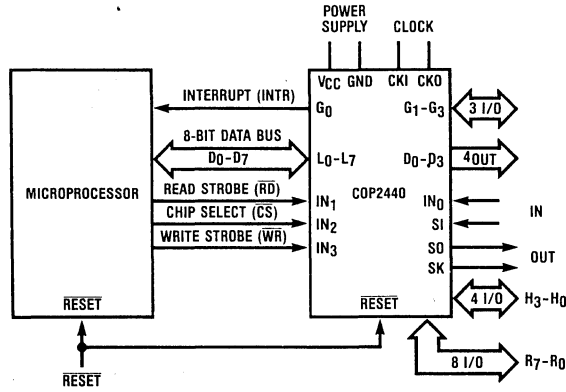


Figure 8. MICROBUS™ Option Interconnect

**Zero-Crossing Detection**  
(not available on the COP2442, COP2342)

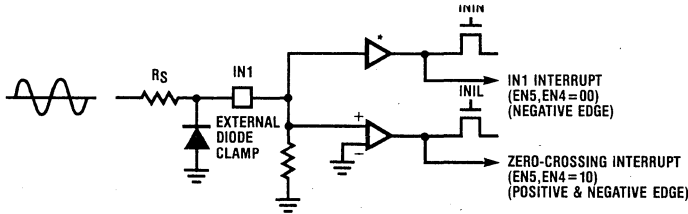
The following features are associated with the IN<sub>1</sub> pin: ININ and INIL instructions input the state of IN<sub>1</sub> to A<sub>1</sub>; IN<sub>1</sub> interrupt generates an interrupt pulse when a low-going transition ("1" to "0") occurs on IN<sub>1</sub>; zero-crossing interrupt generates an interrupt pulse when an IN<sub>1</sub> transition occurs (both "1" to "0" and "0" to "1").

If the zero-crossing detector is mask-programmed in (see Figure 9a), the INIL instruction and zero-crossing interrupt will input the state of IN<sub>1</sub> through the true zero-crossing detector ("1" if input > 0V, "0" if input < 0V). The ININ instruction and IN<sub>1</sub> interrupt will then have unique logic HIGH and LOW levels depending on the IN port input level chosen. If normal (TTL) level is chosen, logic HIGH level is 3.0V (3.3V for COP2340/2341) and logic LOW level is 0.8V (0.6V for COP2340/2341); if high trip level is chosen, logic HIGH level is 5.4V and logic LOW level is 1.2V. If the zero-crossing detector is not mask-programmed in (see Figure 9b), IN<sub>1</sub> will have logic

HIGH and LOW levels that are defined for the IN port (see option list).

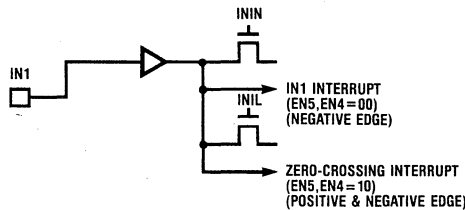
The zero-crossing detector input contains a small hysteresis (50mV typical) to eliminate signal noise, and is not a high impedance input but contains a resistive load to ground. Since this input can withstand a voltage range of -0.8V to +12V, an external clamping diode is needed for most input signals, as shown in Figure 9a, to limit the voltage below ground. An external resistor, R<sub>S</sub> may be needed for the following two cases:

- Input signal exceeds 12V; R<sub>S</sub> and the internal resistor act as a voltage divider to reduce the voltage at the input pin to below 12V.
- Signal comes from a low impedance source; when the voltage at the pin is clamped to -0.7V by the forward bias voltage of an external diode, R<sub>S</sub> limits the current going through the diode.



\*NOTE: THIS INPUT HAS A DIFFERENT SET OF LOGIC HIGH AND LOW LEVELS; SEE ABOVE DESCRIPTION

a. Zero-Crossing Detect Logic Option



b. IN<sub>1</sub> without Zero-Crossing Detect Logic

Figure 9. IN<sub>1</sub> Mask-Programmable Options

### Initialization

The reset logic, internal to the COP2440, will initialize the device upon power-up if the power supply rise time is less than 1ms and greater than 1 $\mu$ s. If the power supply rise time is greater than 1ms, the user must provide an external RC network and diode to the RESET pin as in Figure 10. The RESET pin is configured as a Schmitt trigger input. If not used, it should be connected to V<sub>CC</sub>. Initialization will occur whenever a logic "0" is applied to the RESET input, provided it stays low for at least three instruction cycle times.

Upon initialization, the PC register is cleared to 0 (ROM address 0) and the A, B, C, D, EN, G, H, IL, L, N, Q, R, and T registers are cleared. The SK output is enabled as a SYNC output by setting the SKL latch, thus providing a clock. RAM (data memory and stack) is not cleared. The first instruction at address 0 must be a CLRA.

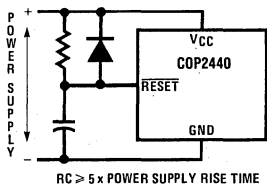


Figure 10. Power-Up Clear Circuit

### Oscillator

There are three basic clock oscillator configurations available, as shown by figure 11.

- a. **Crystal Controlled Oscillator.** CKI and CKO are connected to an external crystal. The execution frequency equals the crystal frequency divided by 16 (optional by 8). Thus a 4 MHz crystal with the divide-by-16 option selected will give a 250 kHz execution frequency (4  $\mu$ s execution time) and a 125 kHz instruction cycle frequency (8  $\mu$ s instruction cycle time).

- b. **External Oscillator.** CKI is an external clock input signal. The external frequency is divided by 16 (optional by 8 or 4) to give the execution frequency. If the divide-by-4 option is selected, the CKI input level is the Schmitt-trigger level. CKO is now available to be used as the RAM power supply (V<sub>R</sub>) or as a general purpose input.

- c. **RC Controlled Oscillator.** CKI is configured as a single pin RC controlled Schmitt trigger oscillator. The execution frequency equals the oscillation frequency divided by 4. CKO is available for non-timing functions.

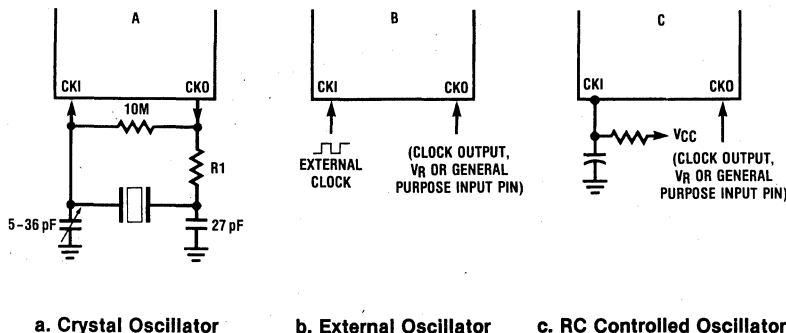
### CKO Pin Options

As an option, CKO can be an oscillator output. In a crystal controlled oscillator system, this signal is used as an output to the crystal network. As another option, CKO can be an interrupt input or a general purpose input, reading into bit 2 of A (accumulator) through the INIL instruction. As another option, CKO can be a RAM power supply pin (V<sub>R</sub>), allowing its connection to a standby/backup power supply to maintain the data integrity of RAM registers 0-3 with minimum power drain when the main supply is inoperative or shut down to conserve power. Using either of the two latter options is appropriate in applications where the system configuration does not require use of the CKO pin for timing functions.

### RAM Keep-Alive Option

Selecting CKO as the RAM power supply (V<sub>R</sub>) allows the user to shut off the chip power supply (V<sub>CC</sub>) and maintain data in the lower 4 registers of the RAM. To insure that RAM data integrity is maintained, the following conditions must be met:

1. RESET must go low before V<sub>CC</sub> goes below spec during power-off; V<sub>CC</sub> must be within spec before RESET goes high on power-up.
2. When V<sub>CC</sub> is on, V<sub>R</sub> must be within the operating voltage range of the chip, and within 1 volt of V<sub>CC</sub>.
3. V<sub>R</sub> must be  $\geq$  3.3V with V<sub>CC</sub> off.



a. Crystal Oscillator

b. External Oscillator

c. RC Controlled Oscillator

### Crystal Oscillator

Crystal Value	R <sub>1</sub>
4 MHz	1k
3.58 MHz	1k
2.10 MHz	2k

### RC Controlled Oscillator

R (k $\Omega$ )	C (pF)	Instruction Execution Time ( $\mu$ s)
13	100	5.0 $\pm$ 20%
6.8	220	5.3 $\pm$ 23%
8.2	300	8.0 $\pm$ 22%
22	100	8.2 $\pm$ 17%

Note: 5k $\Omega$   $\leq$  R  $\leq$  50k $\Omega$   
50 pF  $\leq$  C  $\leq$  360 pF

Figure 11. COP2440/2441/2442 Oscillators

## I/O Options

COP2440 inputs have the following optional configurations, illustrated in figure 12:

- An on-chip depletion load device to  $V_{CC}$ .
- A Hi-Z input which must be driven to a "1" or "0" by external components.
- A resistive load to GND for the zero-crossing input option ( $IN_1$  only).

COP2440 outputs have the following optional configurations:

- Standard** — an enhancement mode device to ground in conjunction with a depletion-mode device to  $V_{CC}$ , compatible with TTL and CMOS input requirements. Available on SO, SK, D, G, and H outputs.
- Open-Drain** — an enhancement-mode device to ground only, allowing external pull-up as required by the user's application. Available on SO, SK, D, G, L, H, and R outputs.
- Push-Pull** — An enhancement-mode device to ground in conjunction with a depletion-mode device paralleled by an enhancement-mode device to  $V_{CC}$ . This configuration has been provided to allow for fast rise and fall times when driving capacitive loads. Available on SO and SK outputs only.
- Standard L,R** — same as d., but may be disabled. Available on L and R outputs only (disabled on reset).
- LED Direct Drive** — an enhancement-mode device to ground and  $V_{CC}$  together with a depletion device to  $V_{CC}$  meeting the typical current sourcing requirements of the segments of an LED display. The sourcing devices are clamped to limit current flow. These devices may be turned off under program control (See Functional Description, EN Register), placing the output in a high-impedance state to provide required LED segment blanking for a multiplexed display. Available on L outputs only.

## Notes:

- When the driver is disabled, the depletion device may cause the output to settle down to an intermediate level between  $V_{CC}$  and GND. This voltage cannot be relied upon as a "1" level when reading the L inputs. The external signal must drive it to a "1" level.
  - Much power is dissipated by this driver in driving an LED. Care must be taken to limit the power dissipation of the chip to within the absolute maximum ratings specified.
- TRI-STATE® Push-Pull** — an enhancement-mode device to ground and  $V_{CC}$ . These outputs are TRI-STATE outputs, allowing for connection of these outputs to a data bus shared by other bus drivers. Available on L and R outputs only (in TRI-STATE mode on reset).
  - Push-Pull R** — same as f., but may be disabled. Available on R outputs only.
  - Additional depletion pull-up** — a depletion load to  $V_{CC}$  with the same current sourcing capability as the input load a., in addition to the output drive chosen. Available on L and R outputs only. *This device cannot be disabled*; therefore, open-drain outputs with "1" output and TRI-STATE outputs do not show high-impedance characteristics. This device is useful in applications where a pull-up with low source current is desired, e.g., reading keyboards and switches.

The above input and output configurations share common enhancement-mode and depletion-mode devices. Specifically, all configurations use one or more of six devices (numbered 1-6 respectively). Minimum and maximum current ( $I_{OUT}$  and  $V_{OUT}$ ) curves are given in Figures 13 and 14 for each of these devices to allow the designer to effectively use these I/O configurations in designing a COP2440 system.

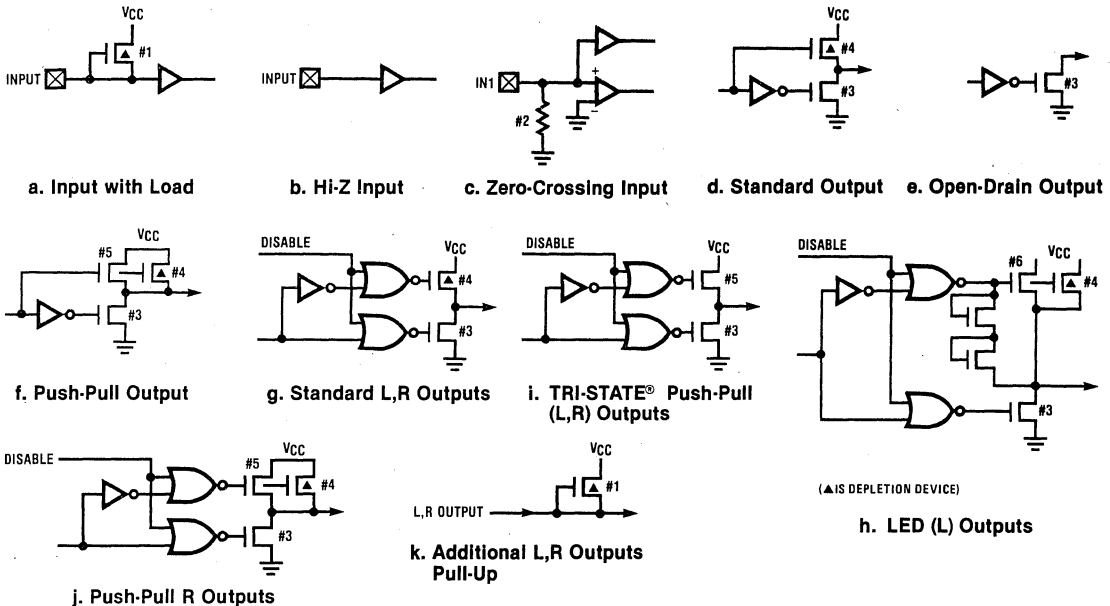
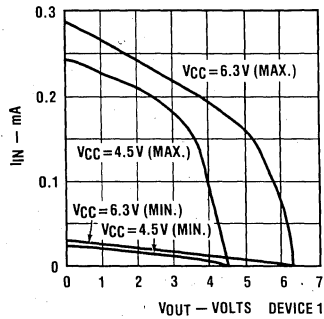
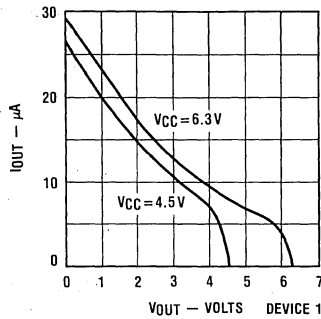


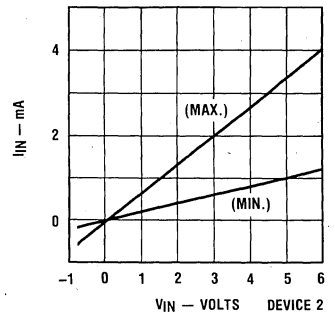
Figure 12. Input/Output Configurations



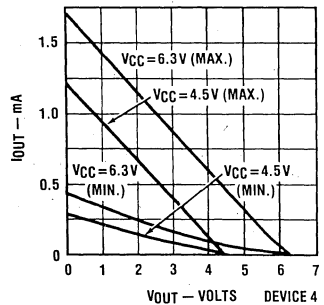
a. Input Load Source Current



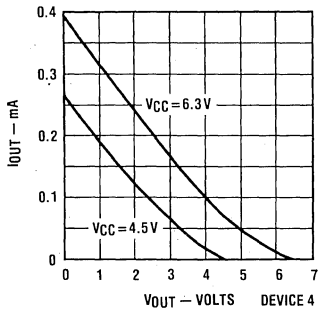
b. Input Load Minimum Source Current



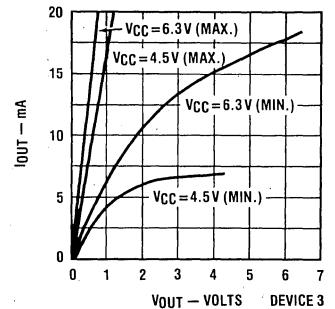
c. Zero-Crossing Detect Input Current



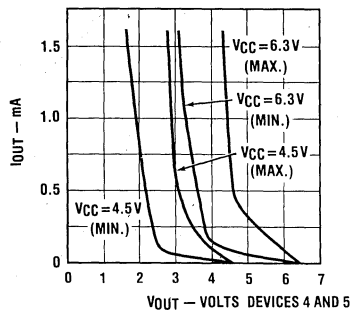
d. Standard Output Source Current



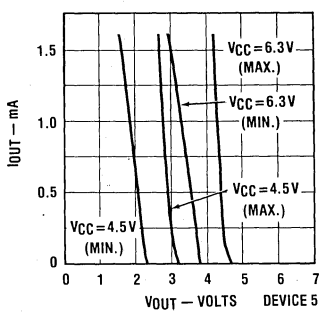
e. Standard Output Minimum Source Current



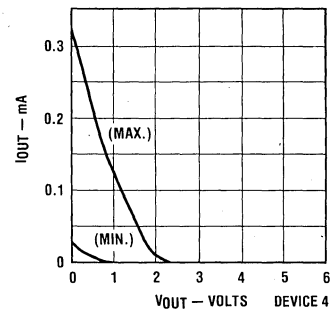
f. Output Sink Current



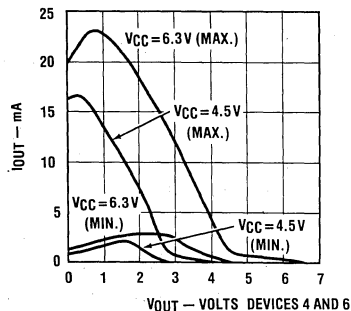
g. Push-Pull Source Current



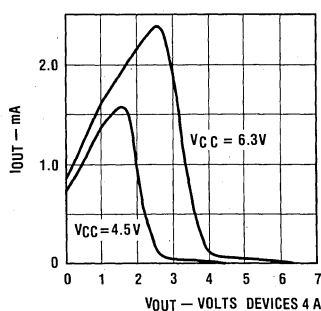
h. TRI-STATE Output Source Current



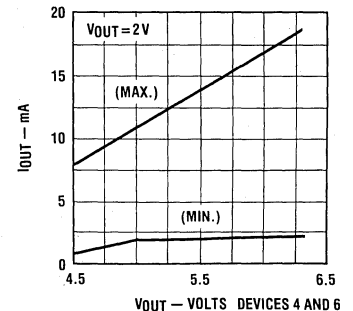
i. Depletion Load OFF Current



j. LED Output Source Current

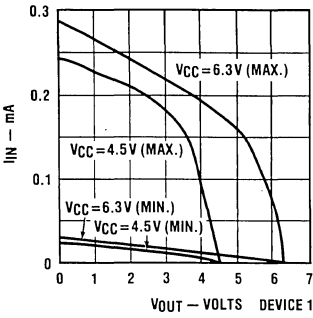


k. LED Output Minimum Source Current

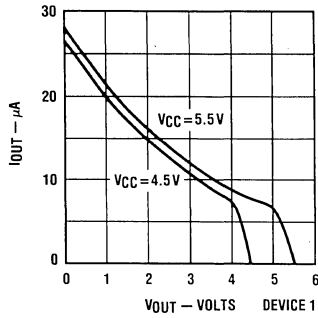


l. LED Output Direct LED Drive

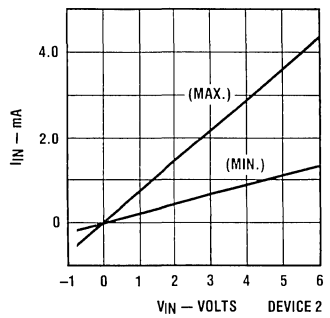
Figure 13. COP2440/2441/2442 I/O Characteristics



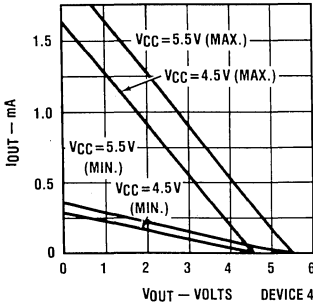
a. Input Load Source Current



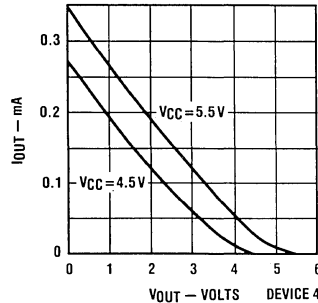
b. Input Load Minimum Source Current



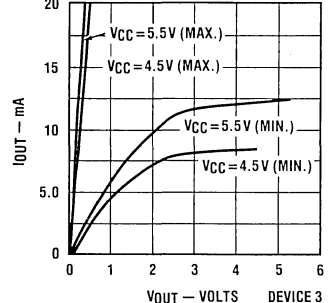
c. Zero-Crossing Detect Input Current



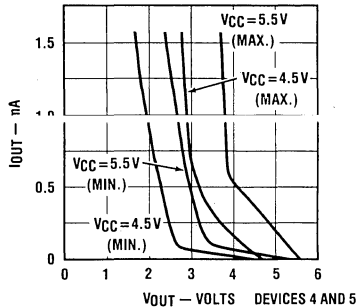
d. Standard Output Source Current



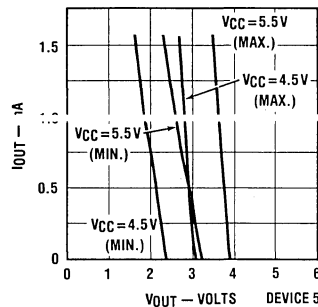
e. Standard Output Minimum Source Current



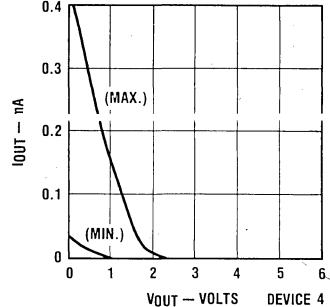
f. Output Sink Current



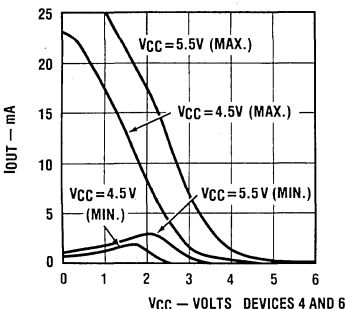
g. Push-Pull Source Current



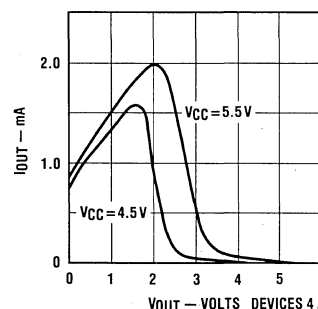
h. TRI-STATE Output Source Current



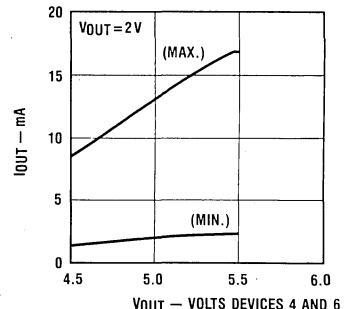
i. Depletion Load OFF Current



j. LED Output Source Current



k. LED Output Minimum Source Current



l. LED Output Direct LED Drive

Figure 14. COP2340/2341/2342 I/O Characteristics



## Power Dissipation

In order not to damage the device by exceeding the absolute maximum power dissipation rating, the amount of power dissipated inside the chip must be carefully controlled. As an example, an application uses a COP2440 in a room temperature (25°C) environment with a  $V_{CC}$  power supply of 6V; IN and SI inputs have internal loads; G and D ports drive loads that may sink up to 2mA into the chip; H port with standard output option reads switches; L port with the LED option drives a multiplexed seven-segment display; R, SO and SK drive MOS inputs that do not source or sink any current.

- At 25°C, maximum power dissipation allowed = 750mW.
- Power dissipation by chip except I/O =  $I_{CC} \times V_{CC} = 35\text{mA} \times 6\text{V} = 210\text{mW}$ .
- Maximum power dissipation by IN, SI =  $5 \times 0.3\text{mA} \times 6\text{V} = 9\text{mW}$
- G and D ports are sinking current from external loads; maximum output voltage with 2mA sink current is less than 0.4V. Power dissipation by G and D ports =  $2\text{mA} \times 0.4\text{V} \times 8 = 6.4\text{mW}$
- Maximum power dissipation by H port =  $4 \times 1.5\text{mA} \times 6\text{V} = 36\text{mW}$
- When the seven segments of the LED are turned on, the output voltage is about 2V, so that the segment current is 17mA. Power dissipation by L port =  $7 \times 17\text{mA} \times (6\text{V} - 2\text{V}) = 476\text{mW}$

This power dissipation caused by driving LEDs is usually the highest among the various sources.

g. R, SO, and SK do not dissipate any significant amount of power because they do not need to source or sink any current.

Total power dissipation (TPD) inside the device is the sum of items b through g above.

$$\text{TPD} = 210 + 9 + 6 + 36 + 476\text{mW} = 737\text{mW}$$

This is within the 750mW limit at room temperature. If this application has to operate at 70°C, then the power dissipation must be reduced to meet the limit at that temperature. Some ways to achieve this would be to limit the LED current or to use an external LED driver.

At 70°C the absolute maximum power dissipation rating drops to 400mW. The user must be careful not to exceed this value.

## COP2440 Series Devices

If the COP2440 is bonded as a 28- or 24-pin device, it becomes the COP2441 or COP2442, respectively, as illustrated in Figure 4. Note that the COP2441 and COP2442 do not include H and R ports. In addition, the COP2442 does not include IN inputs; use of this option precludes the use of the IN options, the interrupt feature with IN as input, the zero-crossing detect option, IN<sub>2</sub> external event counter input, and the MICROBUSTM option. All other options are available.

COP2340, COP2341, and COP2342 are extended temperature versions of the COP2440, COP2441, and COP2442, respectively.

## COP2440 Series Instruction Set

Table 2 is a symbol table providing internal architecture, instruction operand and operation symbols used in the instruction set table.

Table 3 provides the mnemonic, operand, machine code, data flow, skip conditions and description associated with each instruction in the COP2440 series instruction set.

Table 2. COP2440 Series Instruction Set Table Symbols

Symbol	Definition	Symbol	Definition
INTERNAL ARCHITECTURE SYMBOLS		INSTRUCTION OPERAND SYMBOLS	
A	4-bit Accumulator	d	4-bit Operand Field, 0-15 binary (RAM Digit Select)
B	8-bit RAM Address Register	r	4-bit Operand Field, 0-9 binary (RAM Register Select)
Br	Upper 4 bits of B (register address)	a	11-bit Operand Field, 0-2047 binary (ROM Address)
Bd	Lower 4 bits of B (digit address)	y	4-bit Operand Field, 0-15 binary (Immediate Data)
C	1-bit Carry Register	RAM(s)	Content of RAM location addressed by s
D	4-bit Data Output Port	RAM <sub>N</sub>	Content of RAM location addressed by stack pointer N
EN	8-bit Enable Register	ROM(t)	Content of ROM location addressed by t
G	4-bit Register to latch data for G I/O Port	OPERATIONAL SYMBOLS	
H	4-bit Register to latch data for H I/O Port	+	Plus
IL	Two 1-bit Latches associated with the IN <sub>3</sub> or IN <sub>0</sub> Inputs	-	Minus
IN	4-bit Input Port	→	Replaces
IN <sub>1</sub> Z	Zero-Crossing Input	↔	Is exchanged with
L	8-bit TRI-STATE® I/O Port	=	Is equal to
M	4-bit contents of RAM Memory pointed to by B Register	$\bar{A}$	The one's complement of A
N	2-bit subroutine return address stack pointer	⊕	Exclusive-OR
PC	11-bit ROM Address Register (program counter)	:	Range of values
Q	8-bit Register to latch data for L I/O Port	V	OR
R	8-bit Register to latch data for R TRI-STATE I/O Port		
SIO	4-bit Shift Register and Counter		
SK	Logic-Controlled Clock Output		
T	8-bit Binary Counter Register		
X	First On-Chip Processor		
Y	Second On-Chip Processor		

Table 3. COP2440 Series Instruction Set

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>ARITHMETIC/LOGIC INSTRUCTIONS</b>						
ASC		30	$\boxed{0011 0000}$	$A + C + RAM(B) \rightarrow A$ Carry $\rightarrow C$	Carry	Add with Carry, Skip on Carry
ADD		31	$\boxed{0011 0001}$	$A + RAM(B) \rightarrow A$	None	Add RAM to A
ADT		4A	$\boxed{0100 1010}$	$A + 10_{10} \rightarrow A$	None	Add Ten to A
AISC	y	5-	$\boxed{0101 y}$	$A + y \rightarrow A$	Carry	Add Immediate, Skip on Carry (y $\neq$ 0)
CASC		10	$\boxed{0001 0000}$	$\bar{A} + RAM(B) + C \rightarrow A$ Carry $\rightarrow C$	Carry	Complement and Add with Carry, Skip on Carry
CLRA		00	$\boxed{0000 0000}$	$0 \rightarrow A$	None	Clear A
COMP		40	$\boxed{0100 0000}$	$\bar{A} \rightarrow A$	None	One's complement of A to A
NOP		44	$\boxed{0100 0100}$	None	None	No Operation
OR		33 1A	$\boxed{0011 0011}$ $\boxed{0001 1010}$	$A \vee M \rightarrow A$	None	OR RAM with A
RC		32	$\boxed{0011 0010}$	"0" $\rightarrow C$	None	Reset C
SC		22	$\boxed{0010 0010}$	"1" $\rightarrow C$	None	Set C
XOR		02	$\boxed{0000 0010}$	$A \oplus RAM(B) \rightarrow A$	None	Exclusive-OR RAM with A
<b>TRANSFER OF CONTROL INSTRUCTIONS</b>						
JID		FF	$\boxed{1111 1111}$	ROM (PC <sub>10:8</sub> , A,M) $\rightarrow$ PC <sub>7:0</sub>	None	Jump Indirect (Note 3)
JMP	a	6- --	$\boxed{0110 0 a_{10:8}}$ $\boxed{a_{7:0}}$	a $\rightarrow$ PC	None	Jump
JR	a	--	$\boxed{1 a_{6:0}}$ (pages 2,3 only) or $\boxed{11 a_{5:0}}$ (all other pages)	a $\rightarrow$ PC <sub>6:0</sub>	None	Jump within Page (Note 4)
JSRP	a	--	$\boxed{10 a_{5:0}}$	PC + 1 $\rightarrow$ RAM <sub>N</sub> N + 1 $\rightarrow$ N 00010 $\rightarrow$ PC <sub>10:6</sub> a $\rightarrow$ PC <sub>5:0</sub>	None	Jump to Subroutine Page (Note 5)
JSR	a	6- --	$\boxed{0110 1 a_{10:8}}$ $\boxed{a_{7:0}}$	PC + 1 $\rightarrow$ RAM <sub>N</sub> N + 1 $\rightarrow$ N a $\rightarrow$ PC	None	Jump to Subroutine
RET		48	$\boxed{0100 1000}$	N - 1 $\rightarrow$ N RAM <sub>N</sub> $\rightarrow$ PC	None	Return from Subroutine
RETSK		49	$\boxed{0100 1001}$	N - 1 $\rightarrow$ N RAM <sub>N</sub> $\rightarrow$ PC	Always Skip on Return	Return from Subroutine then Skip

Table 3. COP2440 Series Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
MEMORY REFERENCE INSTRUCTIONS						
CAME		33	0011 0011	A → EN <sub>7:4</sub>	None	Copy A, RAM to EN (Processor Y loads EN <sub>2</sub> , EN <sub>7</sub> only)
		1F	0001 1111	RAM(B) → EN <sub>3:0</sub>		
CAMQ		33	0011 0011	A → Q <sub>7:4</sub>	None	Copy A, RAM to Q
		3C	0011 1100	RAM(B) → Q <sub>3:0</sub>		
CAMT		33	0011 0011	A → T <sub>7:4</sub>	None	Copy A, RAM to T
		3F	0011 1111	RAM(B) → T <sub>3:0</sub>		
CEMA		33	0011 0011	EN <sub>7:4</sub> → RAM(B)	None	Copy EN to RAM, A
		0F	0000 1111	EN <sub>3:0</sub> → A		
CQMA		33	0011 0011	Q <sub>7:4</sub> → RAM(B)	None	Copy Q to RAM, A
		2C	0010 1100	Q <sub>3:0</sub> → A		
CTMA		33	0011 0011	T <sub>7:4</sub> → RAM(B)	None	Copy T to RAM, A
		2F	0010 1111	T <sub>3:0</sub> → A		
LD	r	-5	00 r 0101 r=0:3	RAM(B) → A Br ⊕ r → Br	None	Load RAM into A, Exclusive-OR Br with r
LDD	r,d	23	00 10 0011	RAM(r,d) → A	None	Load A with RAM pointed to directly by r,d
		--	0 r d r=0:7			
LID		33	0011 0011	ROM(PC <sub>10:8</sub> ,A,M) → M,A	None	Load RAM, A indirect
		19	0001 1001			
LQID		BF	1011 1111	ROM(PC <sub>10:8</sub> ,A,M) → Q	None	Load Q Indirect (Note 3)
RMB	0	4C	0100 1100	0 → RAM(B) <sub>0</sub>	None	Reset RAM Bit
	1	45	0100 0101	0 → RAM(B) <sub>1</sub>		
	2	42	0100 0010	0 → RAM(B) <sub>2</sub>		
	3	43	0100 0011	0 → RAM(B) <sub>3</sub>		
SMB	0	4D	0100 1101	1 → RAM(B) <sub>0</sub>	None	Set RAM Bit
	1	47	0100 0111	1 → RAM(B) <sub>1</sub>		
	2	46	0100 0110	1 → RAM(B) <sub>2</sub>		
	3	4B	0100 1011	1 → RAM(B) <sub>3</sub>		
STII	y	7-	0111 y	y → RAM(B) Bd + 1 → Bd	None	Store Memory Immediate and Increment Bd
X	r	-6	00 r 0110 r=0:3	RAM(B) ↔ A Br ⊕ r → Br	None	Exchange RAM with A, Exclusive-OR Br with r
XAD	r,d	23	0010 0011	RAM(r,d) ↔ A	None	Exchange A with RAM pointed to directly by r,d
		--	1 r d r=0:7			
XDS	r	-7	00 r 0111 r=0:3	RAM(B) ↔ A Bd - 1 → Bd Br ⊕ r → Br	Bd decrements past 0	Exchange RAM with A and Decrement Bd, Exclusive-OR Br with r
XIS	r	-4	00 r 0100 r=0:3	RAM(B) ↔ A Bd + 1 → Bd Br ⊕ r → Br	Bd increments past 15	Exchange RAM with A and Increment Bd, Exclusive-OR Br with r

Table 3. COP2440 Series Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>REGISTER REFERENCE INSTRUCTIONS</b>						
CAB		50	0101 0000	A → Bd	None	Copy A to Bd
CBA		4E	0100 1110	Bd → A	None	Copy Bd to A
LBI	r,d	--	00  r   (d-1) r=0:3,d=0,9:15 or 0011 0011 or 1  r   d r=0:7,any d	r,d → B	Skip until not a LBI	Load B Immediate with r,d (Note 6)
LEI	y	33 6-	0011 0011 0110  y	y → EN <sub>3,0</sub>	None	Load lower half of EN Immediate (Processor Y loads EN <sub>2</sub> only)
XABR		12	0001 0010	A ↔ Br	None	Exchange A with Br
XAN		33 0B	0011 0011 0000 1011	A ↔ N(0,0 → A <sub>3</sub> ,A <sub>2</sub> )	None	Exchange A with N
<b>TEST INSTRUCTIONS</b>						
SKC		20	0010 0000		C = "1"	Skip if C is True
SKE		21	0010 0001		A = RAM(B)	Skip if A Equals RAM
SKGZ		33 21	0011 0011 0010 0001		G <sub>3,0</sub> = 0	Skip if G is Zero (all 4 bits)
SKGBZ		33	0011 0011	1st byte		Skip if G Bit is Zero
	0	01	0000 0001	} 2nd byte	G <sub>0</sub> = 0	
	1	11	0001 0001		G <sub>1</sub> = 0	
	2	03	0000 0011		G <sub>2</sub> = 0	
	3	13	0001 0011		G <sub>3</sub> = 0	
SKMBZ		0 1 2 3	0000 0001 0001 0001 0000 0011 0001 0011		RAM(B) <sub>0</sub> = 0 RAM(B) <sub>1</sub> = 0 RAM(B) <sub>2</sub> = 0 RAM(B) <sub>3</sub> = 0	Skip if RAM Bit is Zero
SKSZ		33 1C	0011 0011 0001 1100		SIO = 0	Skip if SIO is Zero
SKT		41	0100 0001		T counter carry has occurred since last test	Skip on Timer (Note 3)

Table 3. COP2440 Series Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description		
INPUT/OUTPUT INSTRUCTIONS								
CAMR		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	A → R <sub>7,4</sub>	None	Output A, RAM to R Port
	0011	0011						
	3D	<table border="1"><tr><td>0011</td><td>1101</td></tr></table>	0011	1101	RAM(B) → R <sub>3,0</sub>			
0011	1101							
ING		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	G → A	None	Input G Port to A
	0011	0011						
	2A	<table border="1"><tr><td>0010</td><td>1010</td></tr></table>	0010	1010				
0010	1010							
INH		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	H → A	None	Input H Port to A
	0011	0011						
	2B	<table border="1"><tr><td>0010</td><td>1011</td></tr></table>	0010	1011				
0010	1011							
ININ		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	IN → A	None	Input IN Inputs to A (Note 2)
	0011	0011						
	28	<table border="1"><tr><td>0010</td><td>1000</td></tr></table>	0010	1000				
0010	1000							
INIL		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	IL <sub>3</sub> , CKO, IN <sub>1Z</sub> , IL <sub>0</sub> → A	None	Input IL Latches to A (Note 3)
	0011	0011						
	29	<table border="1"><tr><td>0010</td><td>1001</td></tr></table>	0010	1001				
0010	1001							
INL		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	L <sub>7,4</sub> → RAM(B)	None	Input L Port to RAM, A
	0011	0011						
	2E	<table border="1"><tr><td>0010</td><td>1110</td></tr></table>	0010	1110	L <sub>3,0</sub> → A			
0010	1110							
INR		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	R <sub>7,4</sub> → RAM(B)	None	Input R Port to RAM, A
	0011	0011						
	2D	<table border="1"><tr><td>0010</td><td>1101</td></tr></table>	0010	1101	R <sub>3,0</sub> → A			
0010	1101							
OBD		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	Bd → D	None	Output Bd to D Port
	0011	0011						
	3E	<table border="1"><tr><td>0011</td><td>1110</td></tr></table>	0011	1110				
0011	1110							
OGI	y	33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	y → G	None	Output to G Port Immediate
		0011	0011					
5-	<table border="1"><tr><td>0101</td><td>y</td></tr></table>	0101	y					
0101	y							
OMG		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	RAM(B) → G	None	Output RAM to G Port
	0011	0011						
	3A	<table border="1"><tr><td>0011</td><td>1010</td></tr></table>	0011	1010				
0011	1010							
OMH		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	RAM(B) → H	None	Output RAM to H Port
	0011	0011						
	3B	<table border="1"><tr><td>0011</td><td>1011</td></tr></table>	0011	1011				
0011	1011							
XAS		4F	<table border="1"><tr><td>0100</td><td>1111</td></tr></table>	0100	1111	A ↔ SIO, C → SKL	None	Exchange A with SIO (Note 3) Processor X only
0100	1111							

**Note 1:** All subscripts for alphabetical symbols indicate bit numbers unless explicitly defined (e.g., Br and Bd are explicitly defined). Bits are numbered 0 to N where 0 signifies the least significant bit (low-order, right-most bit). For example, A<sub>3</sub> indicates the most significant (left-most) bit of the 4-bit A register.

**Note 2:** The ININ instruction is not available on the 24-pin COP2442/COP2342 since this device does not contain the IN inputs.

**Note 3:** For additional information on the operation of the XAS, JID, LQID, INIL, and SKT instructions, see below.

**Note 4:** The JP instruction allows a jump, while in subroutine pages 2 or 3, to any ROM location within the two-page boundary of pages 2 or 3. The JP instruction, otherwise, permits a jump to a ROM location within the current 64-word page. JP may not jump to the last word of a page.

**Note 5:** A JSRP transfers program control to subroutine page 2 (00010 is loaded into the upper 5 bits of P). A JSRP may not be used when in pages 2 or 3. JSRP may not jump to the last word in page 2.

**Note 6:** LBI is a single-byte instruction if d = 0, 9, 10, 11, 12, 13, 14, or 15. The machine code for the lower 4 bits equals the binary value of the "d" data *minus 1*, e.g., to load the lower four bits of B (Bd) with the value 9 (1001<sub>2</sub>), the lower 4 bits of the LBI instruction equal 8 (1000<sub>2</sub>). To load 0, the lower 4 bits of the LBI instruction should equal 15 (1111<sub>2</sub>).

## Description of Selected Instructions

The following information is provided to assist the user in understanding the operation of several unique instructions and to provide notes useful to programmers in writing COP2440 programs.

### XAS Instruction

XAS (Exchange A with SIO) can only be executed by processor X. It exchanges the 4-bit contents of the accumulator with the 4-bit contents of the SIO register. The contents of SIO will contain serial-in/serial-out shift register or binary counter data, depending on the value of the EN register. An XAS instruction will also affect the SK output. (See Functional Description, EN register, above). If SIO is selected as a shift register, an XAS instruction must be performed once every 4 instruction cycles to effect a continuous data stream. Processor Y treats XAS as NOP.

### JID Instruction

JID (Jump Indirect) is an indirect addressing instruction, transferring program control to a new ROM location pointed to indirectly by A and M. It loads the lower 8 bits of the ROM address register PC with the contents of ROM addressed by the 11-bit word,  $PC_{10:8}$ , A, M.  $PC_{10}$ ,  $PC_9$  and  $PC_8$  are not affected by this instruction.

Note that JID requires 2 instruction cycles if executed, 1 instruction cycle time if skipped.

### INIL Instruction

INIL (Input IL Latches to A) inputs 2 latches,  $IL_3$  and  $IL_0$ , CKO and  $IN_1$  into A (see Figure 15). The  $IL_3$  and  $IL_0$  latches are set if a low-going pulse ("1" to "0") has occurred on the  $IN_3$  and  $IN_0$  inputs since the last INIL instruction, provided the input pulse stays low for at least one instruction cycle. Execution of an INIL inputs  $IL_3$  and  $IL_0$  into A3 and A0 respectively, and resets these latches to allow them to respond to subsequent low-going pulses on the  $IN_3$  and  $IN_0$  lines. If CKO is mask-programmed as a general purpose input, an INIL will input the state of CKO into A2. If CKO has not been so programmed, a "1" will be placed in A2. Unlike the COP420/420C/420L/444L series, INIL will input  $IN_1$  into A1. If zero-crossing detect is selected, the  $IN_1$  input will go through the detection logic, thus allowing the user to interrogate the input,

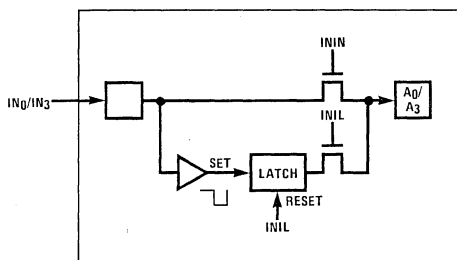


Figure 15. INIL Hardware Implementation

sending a "1" if the input is above zero volts and a "0" if it is below zero volts. INIL is useful in recognizing pulses of short duration or pulses which occur too often to be read conveniently by an ININ instruction. It is also useful in checking the status of the zero-crossing detect input. The general purpose input  $IN_3$ - $IN_0$  are input to A upon execution of an ININ instruction, and the  $IN_1$  input does not go through zero-crossing logic so that it has the same logic level as the other IN inputs for the ININ instruction (see Figure 9).

Note: IL latches are cleared on reset. This is different from the COP420/420C/420L/444L series.

### LQID Instruction

LQID (Load Q Indirect) loads the 8-bit Q register with the contents of ROM pointed to by the 11-bit word  $PC_{10:PC_8}$ , A, M. LQID can be used for table lookup or code conversion such as BCD to seven-segment. Note that LQID takes two instruction cycles if executed and one instruction cycle if skipped. Unlike most other COPSTM processors, this instruction does not push the stack.

### LID Instruction

LID (Load Indirect) loads M and A with the contents of ROM pointed to by the 11-bit word  $PC_{10:PC_8}$ , A, M. Note that LID takes three instruction cycles if executed and two if skipped.

### SKT Instruction

The SKT (Skip On Timer) instruction tests the state of the T counter (see internal logic, above) overflow latch, executing the next program instruction if the latch is not set. If the latch has been set since the previous test, the next program instruction is skipped and the latch is reset. The features associated with this instruction allow the processor to generate its own time-base for real-time processing, rather than relying on an external input signal.

### Instruction Set Notes

- The first word of a COP2440 program (ROM address 0) must be a CLRA (Clear A) instruction.
- Although skipped instructions are not executed, they are still fetched from program memory. Thus program paths take the same number of cycle times whether instructions are skipped or executed, except for LID, LQID, and JID.
- The ROM is organized into 32 pages of 64 words each. The Program Counter is an 11-bit binary counter, and will count through page boundaries. If a JP, JSRP, JID, LQID, or LID instruction is the last word of a page, the instruction operates as if it were in the next page. For example: a JP located in the last word of a page will jump to a location in the next page. Also, a LQID or JID located in the last word of page 3, 7, 11, 15, 19, 23, 27, or 31 will access data in the next group of four pages.

## Option List

The COP2440 mask-programmable options are assigned numbers which correspond with the COP2440 pins.

### Option 1: L<sub>1</sub> I/O Port (see note below)

- = 0: Standard output
- = 1: Open-drain output
- = 2: LED direct drive output
- = 3: TRI-STATE<sup>®</sup> output
- = 4: same as 0 with extra load device to V<sub>CC</sub>
- = 5: same as 1 with extra load device to V<sub>CC</sub>
- = 6: same as 2 with extra load device to V<sub>CC</sub>
- = 7: same as 3 with extra load device to V<sub>CC</sub>

### Option 2: L<sub>0</sub> I/O Port

same as Option 1

### Option 3: SI Input

- = 0: Input with load device to V<sub>CC</sub>
- = 1: Hi-Z input

### Option 4: SO Output

- = 0: Standard output
- = 1: Open-drain output
- = 2: Push-pull output

### Option 5: SK Output

same as Option 4

### Option 6: IN<sub>0</sub> Input

same as Option 3

### Option 7: IN<sub>3</sub> Input

same as Option 3

### Option 8: G<sub>0</sub> I/O Port

- = 0: Standard output
- = 1: Open-drain output

### Option 9: G<sub>1</sub> I/O Port

same as Option 8

### Option 10: G<sub>2</sub> I/O Port

same as Option 8

### Option 11: G<sub>3</sub> I/O Port

same as Option 8

### Option 12: H<sub>0</sub> I/O Port

same as Option 8

### Option 13: H<sub>1</sub> I/O Port

same as Option 8

### Option 14: H<sub>2</sub> I/O Port

same as Option 8

### Option 15: H<sub>3</sub> I/O Port

same as Option 8

### Option 16: D<sub>3</sub> Output

same as Option 8

### Option 17: D<sub>2</sub> Output

same as Option 8

### Option 18: D<sub>1</sub> Output

same as Option 8

### Option 19: D<sub>0</sub> Output

same as Option 8

### Option 20: GND — No options available

### Option 21: CKO Pin

- = 0: Oscillator output
- = 1: RAM power supply (V<sub>R</sub>) input
- = 2: General purpose input with load device to V<sub>CC</sub>
- = 3: General purpose Hi-Z input

### Option 22: CKI Input

- = 0: Crystal input divided by 16
- = 1: Crystal input divided by 8
- = 2: Single-pin RC controlled oscillator (+ 4)
- = 3: Schmitt trigger clock input (+ 4)

### Option 23: RESET Input

same as Option 3

### Option 24: R<sub>7</sub> I/O Port (see note below)

- = 0: Standard output
- = 1: Open-drain output
- = 2: Push-pull output
- = 3: TRI-STATE<sup>®</sup> output
- = 4: same as 0 with extra load device to V<sub>CC</sub>
- = 5: same as 1 with extra load device to V<sub>CC</sub>
- = 6: same as 2 with extra load device to V<sub>CC</sub>
- = 7: same as 3 with extra load device to V<sub>CC</sub>

### Option 25: R<sub>6</sub> I/O Port

same as Option 24

### Option 26: R<sub>5</sub> I/O Port

same as Option 24

### Option 27: R<sub>4</sub> I/O Port

same as Option 24

### Option 28: R<sub>3</sub> I/O Port

same as Option 24

### Option 29: R<sub>2</sub> I/O Port

same as Option 24

### Option 30: R<sub>1</sub> I/O Port

same as Option 24

### Option 31: R<sub>0</sub> I/O Port

same as Option 24

### Option 32: L<sub>7</sub> I/O Port

same as Option 1

### Option 33: L<sub>6</sub> I/O Port

same as Option 1

### Option 34: L<sub>5</sub> I/O Port

same as Option 1

### Option 35: L<sub>4</sub> I/O Port

same as Option 1

### Option 36: IN<sub>1</sub> Input

- = 0: Input with load device to V<sub>CC</sub>
- = 1: Hi-Z Input
- = 2: Zero-crossing detect input (Option 41 = 0)

### Option 37: IN<sub>2</sub> Input

same as Option 3

**Option List (continued)**

Option 38: L<sub>3</sub> I/O Port  
same as Option 1

Option 39: L<sub>2</sub> I/O Port  
same as Option 1

Option 40: V<sub>CC</sub> — no options available

Option 41: COP Function  
= 0: Normal  
= 1: MICROBUS™ option

Option 42: IN Input Levels  
= 0: Standard TTL input levels ("0" = 0.8V, "1" = 2.0V)  
= 1: Higher voltage input levels ("0" = 1.2V, "1" = 3.6V)

Option 43: G Input Levels  
same as Option 42

Option 44: L Input Levels  
same as Option 42

Option 45: CKO Input Levels  
same as Option 42

Option 46: SI Input Levels  
same as Option 42

Option 47: R Input Levels  
same as Option 42

Option 48: H Input Levels  
same as Option 42

Option 49: No option available

Option 50: COP Bonding  
= 0: COP 2440 (40-pin device)  
= 1: COP2441 (28-pin device)  
= 2: COP2442 (24-pin device)  
= 3: COP 2440 and COP2441  
= 4: COP2440 and COP2442  
= 5: COP2440, COP2441, and COP2442  
= 6: COP2441 and COP2442

**Note on L and R I/O Port Options**

If L and R I/O Ports are used as inputs, the following must be observed:

- Open-Drain output (selection 1) is allowed only if external pull-up is provided.
- If L and R output ports are disabled when reading, an external pull-up is required unless selections 4, 5, 6, or 7 are chosen.
- If L output port is enabled, selections 3 and 7 are not allowed.
- If R output port is enabled, selections 2, 3, 6, and 7 are not allowed.

**Test Mode (Non-Standard Operation)**

The SO output has been configured to provide for standard test procedures for the custom-programmed COP2440. With SO forced to logic "1", two test modes are provided, depending upon the value of SI:

- RAM and Internal Logic Test Mode (SI = 1)
- ROM Test Mode (SI = 0)

These special test modes should not be employed by the user; they are intended for manufacturing test only.

**Application Example**

The dual processors on the COP2440 enable the user to do functions that are not possible (or very difficult) to do on a single processor. Programming is also easier using the dual processor COP2440, because different tasks can be partitioned to each processor. The power of the dual processor becomes apparent when two or more tasks must be performed where one task is constant and cannot be disturbed or interrupted.

The following is a simple example to show the dual processor in action. In this example application, the chip must monitor two switches and two pulse train inputs. It must also output a *continuous* square wave which is a function of all the inputs (see Figure 16).

The tasks are partitioned such that processor Y will read a value in RAM and count it down to toggle an output. This is a constant process that gives a continuous output stream. Processor X counts pulses on one input, measures the period of another, and reads switches. Processor X will be interrupted to do a complex calculation based on the input values.

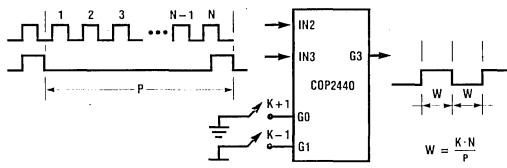


Figure 16. COP2440 Application Example

This is exceedingly difficult to do using a single processor, since the one output must be constantly updated. Therefore, the programmer of a single processor trying to do this task would have to interleave the code to update the output with *all* of the other code (i.e., multiply, divide, add, counting, etc.).

The following is a flow chart of the COP2440 program to do the described tasks. Processor Y reads data register W and counts it down in a fixed loop. Processor X counts pulses on IN<sub>2</sub> in the T counter and measures the period of the pulse on IN<sub>3</sub> by a software loop that counts instruction cycles in data register C. When a negative edge comes on IN<sub>3</sub>, the calculation of pulse width is performed and the two keys are read. The program then branches back to the main loop to start again.

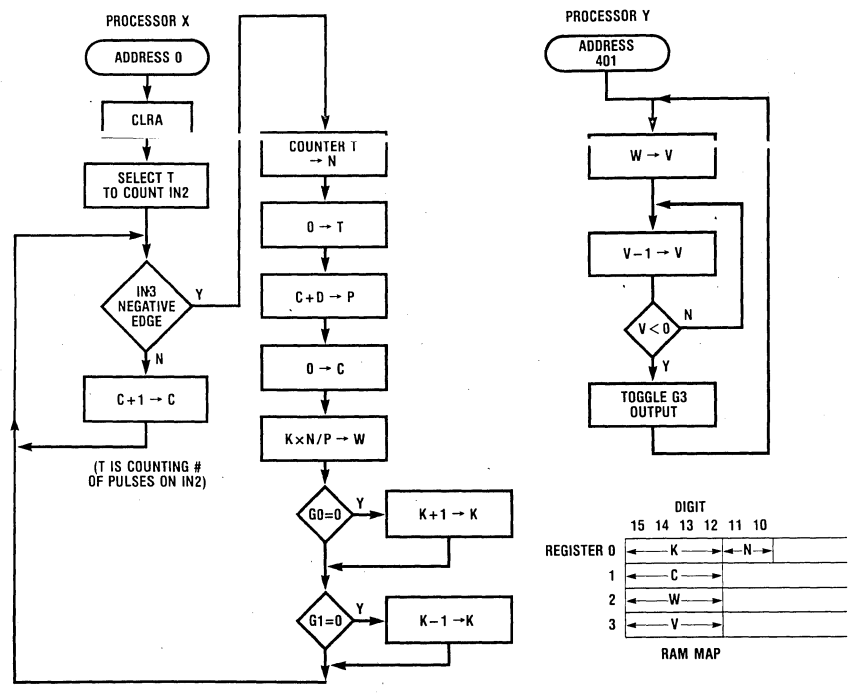




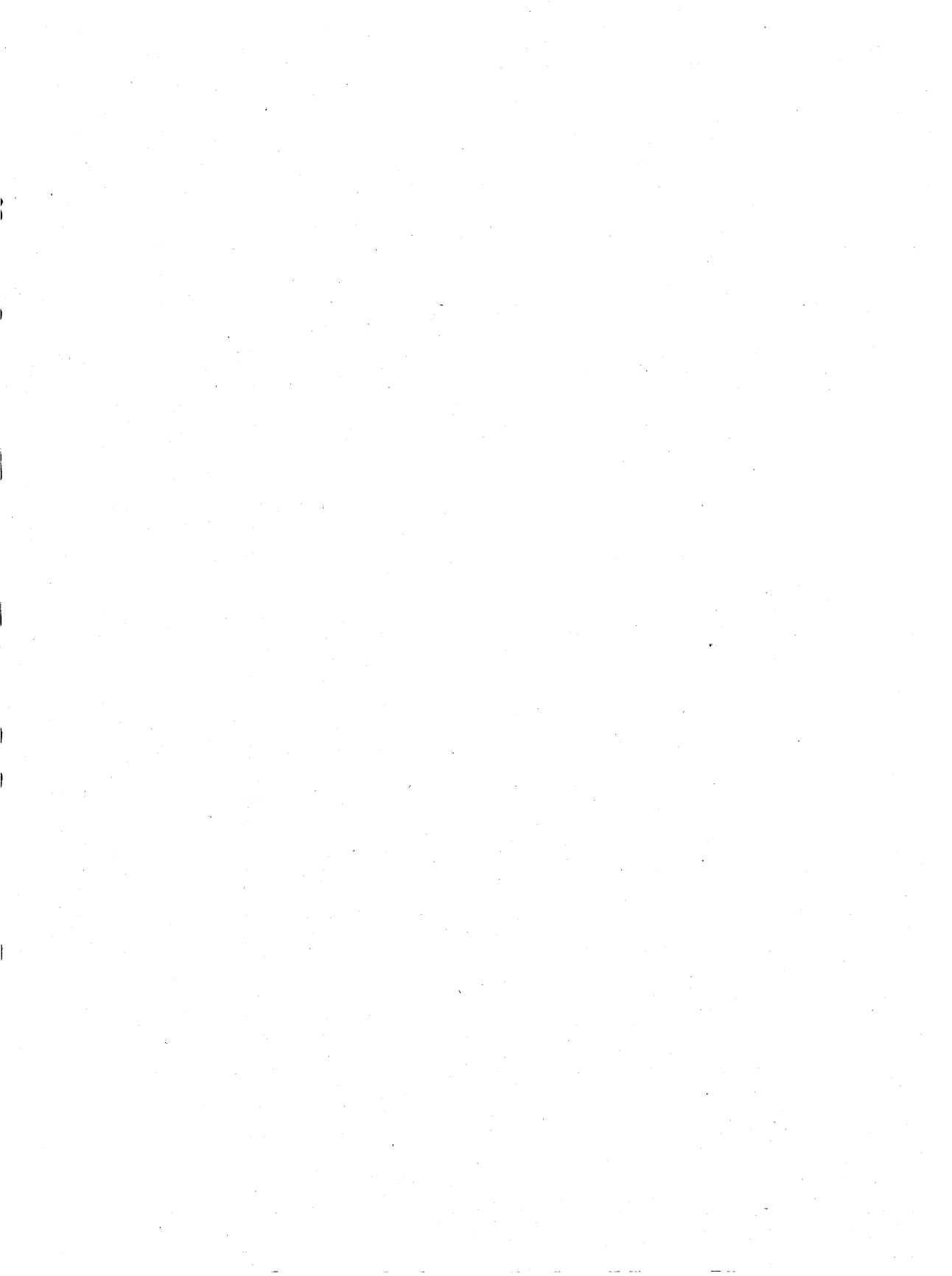
```

;
; DECREMENT REGISTER ROUTINE
;
08B  DECR:  RC                      ; RESET CARRY
08C  DECL:  CLRA                    ;
08D                    CASC          ; M-1 -> A
08E                    NOP
08F                    XIS           ; STORE RESULT
090                    JP          DECL ; LOOP TIL LAST DIGIT
091                    RET
;
; . = 0401                      ; PROCESSOR Y START ADDRESS
;
;
; PROCESSOR Y OUTPUTS SQUARE WAVE TO G3
;
401  YMAIN: OGI          7           ; RESET G3, G0&G1 SET HIGH
403  YML:   LBI         2, 12       ; ADDRESS W DATA REG.
404  XFER:  LD          1           ;
405                    XIS         1 ; THIS LOOP COPIES W -> V
406                    JP          XFER ;
407  YLOOP: LBI         3, 12       ; ADDRESS V DATA REG.
408                    JSRP        DECR ; V-1 -> V
409                    SKC          ; TEST C
40A                    JP          TOGG ; FALSE (= BORROW)
40B                    JP          YLOOP ; TRUE, CONTINUE
;
40C  TOGG:  SKGBZ        3           ; TEST STATE OF G3
40E                    JP          RS  ; TRUE, SO RESET IT
40F                    OGI         15 ; SET G3, KEEP G0&G1 HIGH
411                    JP          YML ; CONTINUE LOOP
412  RS:    JP          YMAIN        ;
;
; END

```



Flow Chart





Section 3  
**ROMless  
Microcontrollers**





# COP401L ROMless N-Channel Microcontroller

## General Description

The COP401L ROMless Microcontroller is a member of the COPS™ family of microcontrollers, fabricated using N-channel, silicon gate MOS technology. The COP401L contains CPU, RAM, I/O and is identical to a COP410L device except the ROM has been removed and pins have been added to output the ROM address and to input the ROM data. In a system the COP401L will perform exactly as the COP410L. This important benefit facilitates development and debug of a COP program prior to masking the final part.

## Features

- Circuit equivalent of COP410L
- Low cost
- Powerful instruction set
- 512 × 8 ROM, 32 × 4 RAM
- Separate RAM power supply pin for RAM keep-alive applications
- Two-level subroutine stack
- 15μs instruction time
- Single supply operation (4.5–9.5V)
- Low current drain (8mA max.)
- Internal binary counter register with serial I/O
- MICROWIRE™ compatible serial I/O
- General purpose outputs
- LSTTL/CMOS compatible in and out
- Direct drive of LED digit and segment lines
- Software/hardware compatible with other members of COP400 family
- Pin-for-pin compatible with COP402 and COP404L

COPS and MICROWIRE are trademarks of National Semiconductor Corp.

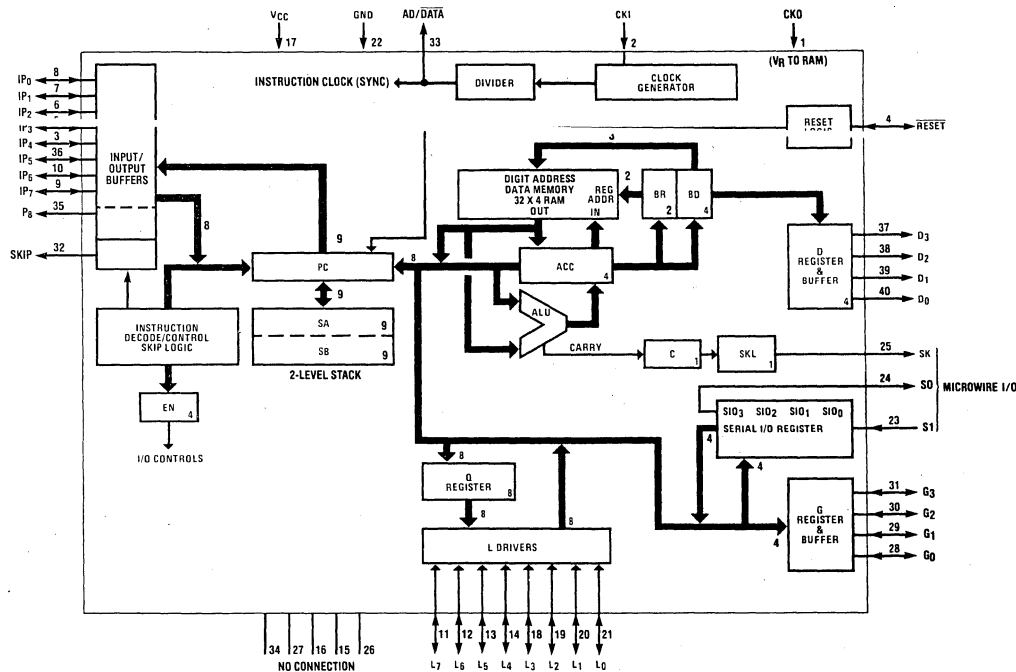


Figure 1. COP401L Block Diagram

**3**

## Absolute Maximum Ratings

Voltage at Any Pin Relative to GND	-0.5V to +10V
Ambient Operating Temperature	0°C to +70°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C
Power Dissipation	0.75 Watt at 25°C 0.4 Watt at 70°C
Total Source Current	120 mA
Total Sink Current	120 mA

*Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

## DC Electrical Characteristics $0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ , $4.5\text{V} \leq V_{CC} \leq 9.5\text{V}$ unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Operating Voltage ( $V_{CC}$ )	(Note 2)	4.5	9.5	V
Power Supply Ripple	peak to peak		0.5	V
Operating Supply Current	all inputs and outputs open		8	mA
Input Voltage Levels				
CKI Input Levels				
Crystal Input				
Logic High ( $V_{IH}$ )		2.0		V
Logic Low ( $V_{IL}$ )		-0.3	0.4	V
RESET Input Levels	Schmitt trigger input			
Logic High		$0.7 V_{CC}$		V
Logic Low		-0.3	0.6	V
IPO-IP7 Input Levels				
Logic High	$V_{CC} = 9.5\text{V}$	2.4		V
Logic High	$V_{CC} = 5\text{V} \pm 5\%$	2.0		V
Logic Low		-0.3	0.8	V
All Other Inputs				
Logic High	$V_{CC} = 9.5\text{V}$	3.0		V
Logic High	$V_{CC} = 5\text{V} \pm 5\%$	2.0		V
Logic Low		-0.3	0.8	V
Input Capacitance			7	pF
Output Voltage Levels				
LSTTL Operation	$V_{CC} = 5\text{V} \pm 5\%$			
Logic High ( $V_{OH}$ )	$I_{OH} = -25\mu\text{A}$	2.7		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 0.36\text{mA}$		0.4	V
IPO-IP7, P8, SKIP	(Note 1)			
Logic Low	$I_{OL} = 1.6\text{mA}$		0.4	V
Output Current Levels				
Output Sink Current				
SO and SK Outputs ( $I_{OL}$ )	$V_{CC} = 9.5\text{V}$ , $V_{OL} = 0.4\text{V}$	1.8		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.9		mA
L <sub>0</sub> -L <sub>7</sub> and G <sub>0</sub> -G <sub>3</sub> Outputs	$V_{CC} = 9.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.8		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.4		mA
D <sub>0</sub> -D <sub>3</sub> Outputs	$V_{CC} = 9.5\text{V}$ , $V_{OL} = 1.0\text{V}$	30		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 1.0\text{V}$	15		mA
CKO				
RAM Power Supply Input	$V_R = 3.3\text{V}$		1.5	mA

**DC Electrical Characteristics** (continued)  $0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{\text{CC}} \leq 9.5\text{V}$  unless otherwise noted.

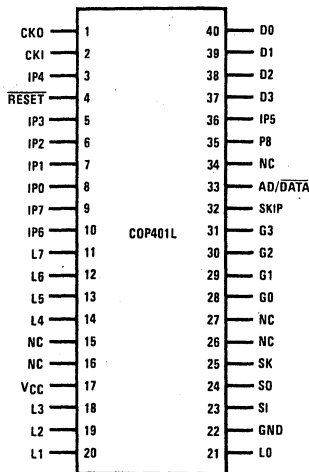
Parameter	Conditions	Min.	Max.	Units
Output Source Current				
D <sub>0</sub> -D <sub>3</sub> , G <sub>0</sub> -G <sub>3</sub> Outputs (I <sub>OH</sub> )	V <sub>CC</sub> = 9.5V, V <sub>OH</sub> = 2.0V	-140	-800	μA
	V <sub>CC</sub> = 4.5V, V <sub>OH</sub> = 2.0V	-30	-250	μA
SO and SK Outputs (I <sub>OH</sub> )	V <sub>CC</sub> = 9.5V, V <sub>OH</sub> = 4.75V	-1.4		mA
	V <sub>CC</sub> = 4.5V, V <sub>OH</sub> = 1.0V	-1.2		mA
L <sub>0</sub> -L <sub>7</sub> Outputs	V <sub>CC</sub> = 9.5V, V <sub>OH</sub> = 2.0V	-3.0	-35	mA
	V <sub>CC</sub> = 6.0V, V <sub>OH</sub> = 2.0V	-3.0	-25	mA
Input Load Source Current (I <sub>IL</sub> )	V <sub>CC</sub> = 5.0V, V <sub>L</sub> = 0V	-10	-140	μA
Total Sink Current Allowed				
All Outputs Combined			120	mA
D Port			100	mA
L <sub>7</sub> -L <sub>4</sub> , G Port			4	mA
L <sub>3</sub> -L <sub>0</sub>			4	mA
All Other Pins			1.8	mA
Total Source Current Allowed				
All I/O Combined			120	mA
L <sub>7</sub> -L <sub>4</sub>			60	mA
L <sub>3</sub> -L <sub>0</sub>			60	mA
Each L Pin			25	mA
All Other Pins			1.5	mA

**AC Electrical Characteristics**  $0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{\text{CC}} \leq 9.5\text{V}$  unless otherwise specified.

Parameter	Conditions	Min.	Max.	Units
Instruction Cycle Time		15	40	μs
CKI				
Input Frequency f <sub>i</sub>	(÷32 mode)	0.8	2.1	MHz
Duty Cycle		30	60	%
Rise Time	f <sub>i</sub> = 2.097 MHz		120	ns
Fall Time			80	ns
INBITE:				
SI, IP7-IP0				
t <sub>SETUP</sub>			2.0	μs
t <sub>HOLD</sub>			1.0	μs
G <sub>3</sub> -G <sub>0</sub> , L <sub>7</sub> -L <sub>0</sub>				
t <sub>SETUP</sub>			8.0	μs
t <sub>HOLD</sub>			1.3	μs
OUTPUT PROPAGATION DELAY	Test Condition: C <sub>L</sub> = 50 pF, V <sub>OUT</sub> = 1.5V			
SO, SK Outputs	R <sub>L</sub> = 20 kΩ		4.0	μs
D <sub>3</sub> -D <sub>0</sub> , G <sub>3</sub> -G <sub>0</sub> , L <sub>7</sub> -L <sub>0</sub>	R <sub>L</sub> = 20 kΩ		5.6	μs
IP7-IP0, P8, SKIP	R <sub>L</sub> = 5 kΩ		7.2	μs
t <sub>pd1</sub> , t <sub>pd0</sub>				

**Note 1:** Pull-up resistors required.**Note 2:** V<sub>CC</sub> voltage change must be less than 0.5V in a 1ms period to maintain proper operation.





Order Number COP401L/N  
NS Package N40A

Figure 2. Connection Diagram

Pin	Description	Pin	Description
L7-L0	8 bidirectional I/O ports with LED segment drive	CKI	System oscillator input
G3-G0	4 bidirectional I/O ports	CKO	RAM power supply input
D3-D0	4 general purpose outputs	RESET	System reset input
SI	Serial input (or counter input)	VCC	Power supply
SO	Serial output (or general purpose output)	GND	Ground
SK	Logic-controlled clock (or general purpose output)	IP7-IP0	8 bidirectional ROM address and data ports
AD/DATA	Address out/data in flag	P8	Most significant ROM address bit output
		SKIP	Instruction skip output

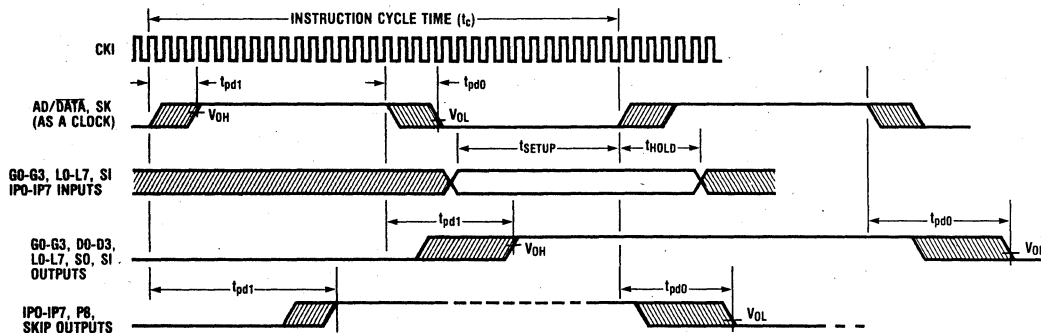


Figure 3. Input/Output Timing Diagram

## Functional Description

A block diagram of the COP401L is given in Figure 1. Data paths are illustrated in simplified form to depict how the various logic elements communicate with each other in implementing the instruction set of the device. Positive logic is used. When a bit is set, it is a logic "1" (greater than 2 volts). When a bit is reset, it is a logic "0" (less than 0.8 volts).

### Program Memory

Program Memory consists of a 512-byte external memory. As can be seen by an examination of the COP401L instruction set, these words may be program instructions, program data or ROM addressing data. Because of the special characteristics associated with the JP, JSRP, JID and LQID instructions, ROM must often be thought of as being organized into 8 pages of 64 words each.

ROM addressing is accomplished by a 9-bit PC register. Its binary value selects one of the 512 8-bit words contained in ROM. A new address is loaded into the PC register during each instruction cycle. Unless the instruction is a transfer of control instruction, the PC register is loaded with the next sequential 9-bit binary count value. Two levels of subroutine nesting are implemented by the 9-bit subroutine save registers, SA and SB, providing a last-in, first-out (LIFO) hardware subroutine stack.

ROM instruction words are fetched, decoded and executed by the Instruction Decode, Control and Skip Logic circuitry.

### Data Memory

Data memory consists of a 128-bit RAM, organized as 4 data registers of 8 4-bit digits. RAM addressing is implemented by a 6-bit B register whose upper 2 bits (Br) select 1 of 4 data registers and lower 3 bits of the 4-bit Bd select 1 of 8 4-bit digits in the selected data register. While the 4-bit contents of the selected RAM digit (M) is usually loaded into or from, or exchanged with, the A register (accumulator), it may also be loaded into the Q latches or loaded from the L ports. RAM addressing may also be performed directly by the XAD 3,15 instruction. The Bd register also serves as a source register for 4-bit data sent directly to the D outputs.

The most significant bit of Bd is not used to select a RAM digit. Hence each physical digit of RAM may be selected by two different values of Bd as shown in Figure 4 below. The skip condition for XIS and XDS instructions will be true if Bd changes between 0 and 15, but NOT between 7 and 8 (see Table 3).

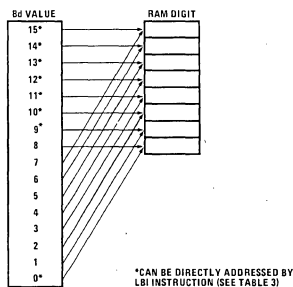


Figure 4. RAM Digit Address to Physical RAM Digit Mapping

### Internal Logic

The 4-bit A register (accumulator) is the source and destination register for most I/O, arithmetic, logic and data memory access operations. It can also be used to load the Bd portion of the B register, to load 4 bits of the 8-bit Q latch data, to input 4 bits of the 8-bit L I/O port data and to perform data exchanges with the SIO register.

A 4-bit adder performs the arithmetic and logic functions of the COP401L, storing its results in A. It also outputs a carry bit to the 1-bit C register, most often employed to indicate arithmetic overflow. The C register, in conjunction with the XAS instruction and the EN register, also serves to control the SK output. C can be outputted directly to SK or can enable SK to be a sync clock each instruction cycle time. (See XAS instruction and EN register description, below.)

The G register contents are outputs to 4 general-purpose bidirectional I/O ports.

The Q register is an internal, latched, 8-bit register, used to hold data loaded from M and A, as well as 8-bit data from ROM. Its contents are output to the L I/O ports when the L drivers are enabled under program control. (See LEI instruction.)

The 8 L drivers, when enabled, output the contents of latched Q data to the L I/O ports. Also, the contents of L may be read directly into A and M. L I/O ports can be directly connected to the segments of a multiplexed LED display (using the LED Direct Drive output configuration option) with Q data being outputted to the Sa-Sg and decimal point segments of the display.

The SIO register functions as a 4-bit serial-in/serial-out shift register or as a binary counter depending on the contents of the EN register. (See EN register description, below.) Its contents can be exchanged with A, allowing it to input or output a continuous serial data stream. SIO may also be used to provide additional parallel I/O by connecting SO to external serial-in/parallel-out shift registers.

The XAS instruction copies C into the SKL Latch. In the counter mode, SK is the output of SKL in the shift register mode, SK outputs SKL ANDed with internal instruction cycle clock.

The EN register is an internal 4-bit register loaded under program control by the LEI instruction. The state of each bit of this register selects or deselects the particular feature associated with each bit of the EN register (EN<sub>3</sub>-EN<sub>0</sub>).

1. The least significant bit of the enable register, EN<sub>0</sub>, selects the SIO register as either a 4-bit shift register or a 4-bit binary counter. With EN<sub>0</sub> set, SIO is an asynchronous binary counter, *decrementing* its value by one upon each low-going pulse ("1" to "0") occurring on the SI input. Each pulse must be at least two instruction cycles wide. SK outputs the value of SKL. The SO output is equal to the value of EN<sub>3</sub>. With EN<sub>0</sub> reset, SIO is a serial shift register shifting left each instruction cycle time. The data present at SI goes into the least significant bit of SIO. SO can be enabled to output the most significant bit of SIO each cycle time. (See 4 below.) The SK output becomes a logic-controlled clock.

2. EN<sub>1</sub> is not used. It has no effect on COP401L operation.

Table 1. Enable Register Modes — Bits EN<sub>3</sub> and EN<sub>0</sub>

EN <sub>3</sub>	EN <sub>0</sub>	SIO	SI	SO	SK
0	0	Shift Register	Input to Shift Register	0	If SKL = 1, SK = Clock If SKL = 0, SK = 0
1	0	Shift Register	Input to Shift Register	Serial Out	If SKL = 1, SK = Clock If SKL = 0, SK = 0
0	1	Binary Counter	Input to Binary Counter	0	If SKL = 1, SK = 1 If SKL = 0, SK = 0
1	1	Binary Counter	Input to Binary Counter	1	If SKL = 1, SK = 1 If SKL = 0, SK = 0

- With EN<sub>2</sub> set, the L drivers are enabled to output the data in Q to the L I/O ports. Resetting EN<sub>2</sub> disables the L drivers, placing the L I/O ports in a high-impedance input state.
- EN<sub>3</sub>, in conjunction with EN<sub>0</sub>, affects the SO output. With EN<sub>0</sub> set (binary counter option selected) SO will output the value loaded into EN<sub>3</sub>. With EN<sub>0</sub> reset (serial shift register option selected), setting EN<sub>3</sub> enables SO as the output of the SIO shift register, outputting serial shifted data each instruction time. Resetting EN<sub>3</sub> with the serial shift register option selected disables SO as the shift register output; data continues to be shifted through SIO and can be exchanged with A via an XAS instruction but SO remains reset to "0." Table I provides a summary of the modes associated with EN<sub>3</sub> and EN<sub>0</sub>.

#### Initialization

The Reset Logic will initialize (clear) the device upon power-up if the power supply rise time is less than 1 ms and greater than 1  $\mu$ s. If the power supply rise time is greater than 1 ms, the user must provide an external RC network and diode to the RESET pin as shown below (Figure 5). The RESET pin is configured as a Schmitt trigger input. If not used it should be connected to V<sub>CC</sub>. Initialization will occur whenever a logic "0" is applied to the RESET input, provided it stays low for at least three instruction cycle times.

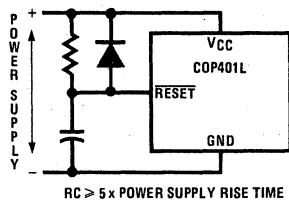


Figure 5. Power-Up Clear Circuit

Upon initialization, the PC register is cleared to 0 (ROM address 0) and the A, B, C, D, EN, and G registers are cleared. The SK output is enabled as a SYNC output, providing a pulse each instruction cycle time. *Data Memory (RAM) is not cleared upon initialization.* The first instruction at address 0 must be a CLRA.

#### External Memory Interface

The COP401L is designed for use with an external Program Memory. This memory may be implemented using any devices having the following characteristics:

- random addressing
- TTL-compatible TRI-STATE<sup>®</sup> outputs
- TTL-compatible inputs
- access time = 5  $\mu$ s max.

Typically these requirements are met using bipolar or MOS PROMs.

During operation, the address of the next instruction is sent out on P8 and IP7 through IP0 during the time that AD/DATA is high (logic "1" = address mode). Address data on the IP lines is stored into an external latch on the high-to-low transition of the AD/DATA line; P8 is a dedicated address output, and does not need to be latched. When AD/DATA is low (logic "0" = data mode), the output of the memory is gated onto IP7 through IP0, forming the input bus. Note that the AD/DATA output has a period of one instruction time, a duty cycle of approximately 50%, and specifies whether the IP lines are used for address output or instruction input.

#### Oscillator

CKI is an external clock input signal. The external frequency is divided by 32 to give the instruction cycle time. The divide-by-32 configuration was chosen to make the COP401L compatible with the COP404L and the COPSTM Development System. However, the +32 configuration is not available on the COP410L/COP411L. It is therefore possible to exactly emulate the system speed (cycle time), but not possible to drive the 401L with the system clock during emulation.

## CKO (RAM Power)

CKO is configured as a RAM power supply pin ( $V_R$ ), allowing its connection to a standby/backup power supply to maintain the integrity of RAM data with minimum power drain when the main supply is inoperative or shut down to conserve power. This pin must be connected to  $V_{CC}$  if the power backup feature is not used. To insure that RAM integrity is maintained, the following conditions must be met:

1.  $\overline{RESET}$  must go low before  $V_{CC}$  goes below spec during power-off;  $V_{CC}$  must be within spec before  $\overline{RESET}$  goes high on power-up.
2. During normal operation,  $V_R$  must be within the operating range of the chip with  $(V_{CC} - 1) \leq V_R \leq V_{CC}$ .
3.  $V_R$  must be  $\geq 3.3V$  with  $V_{CC}$  off.

## Input/Output Configurations

COP401L outputs have the following configurations, illustrated in Figure 6:

- Standard** — an enhancement mode device to ground in conjunction with a depletion-mode device to  $V_{CC}$ , compatible with LSTTL and CMOS input requirements. (Used on D and G outputs.)
- Open-Drain** — an enhancement-mode device to ground only, allowing external pull-up as required by the user's application. (Used on IP, P and SKIP outputs.)
- Push-Pull** — An enhancement-mode device to ground in conjunction with a depletion-mode device paralleled

enhancement-mode device to  $V_{CC}$ . This configuration has been provided to allow for fast rise and fall times when driving capacitive loads. (Used on SO and SK outputs.)

- LED Direct Drive** — an enhancement-mode device to ground and to  $V_{CC}$ , meeting the typical current sourcing requirements of the segments of an LED display. The sourcing device is clamped to limit current flow. These devices may be turned off under program control (See Functional Description, EN Register), placing the outputs in a high-impedance state to provide required LED segment blanking for a multiplexed display. (Used on L outputs.)

COP401L inputs have an on-chip depletion load device to  $V_{CC}$ .

The above input and output configurations share common enhancement-mode and depletion-mode devices. Specifically, all configurations use one or more of five devices (numbered 1-5, respectively). Minimum and maximum current ( $I_{OUT}$  and  $V_{OUT}$ ) curves are given in Figure 7 for each of these devices to allow the designer to effectively use these I/O configurations in designing a system.

An important point to remember is that even when the L drivers are disabled, the depletion load device will source a small amount of current (see Figure 7, Device 2); however, when the L-lines are used as inputs, the disabled depletion device can *not* be relied on to source sufficient current to pull an input to a logic "1".

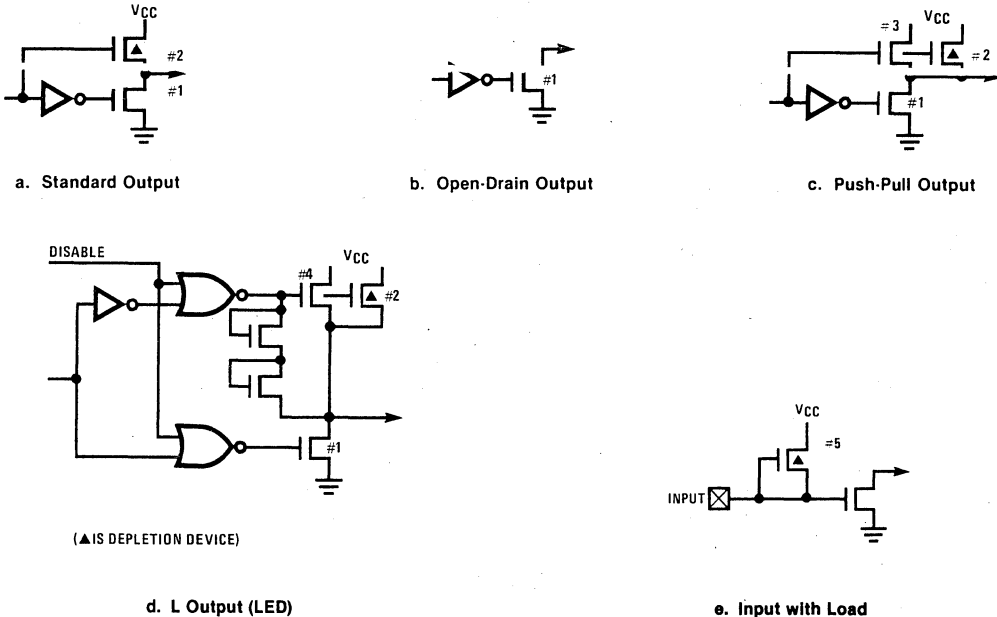


Figure 6. Output Configurations

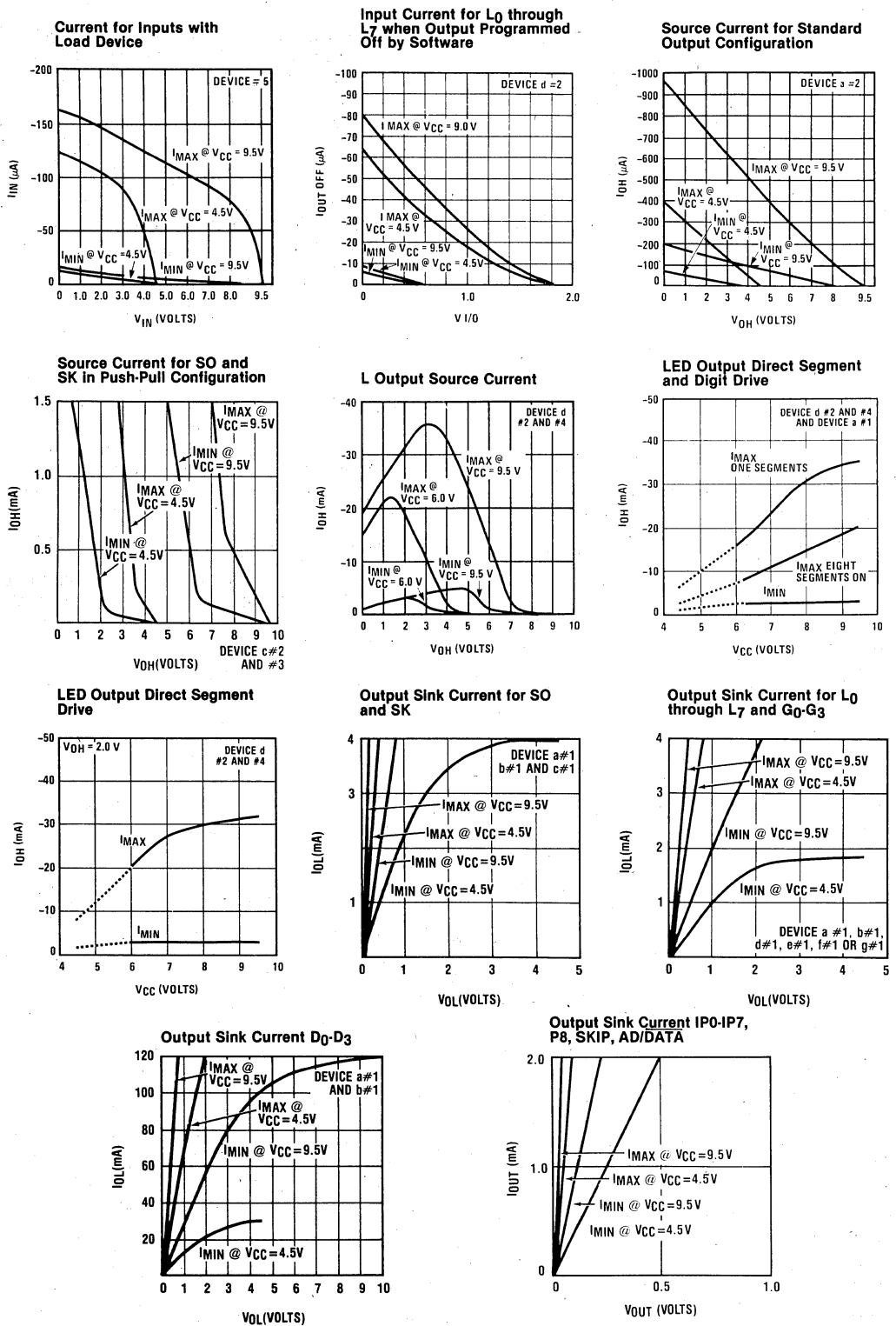


Figure 7. I/O Characteristics

## COP401L Instruction Set

Table 2 is a symbol table providing internal architecture, instruction operand and operational symbols used in the instruction set table.

Table 3 provides the mnemonic, operand, machine code, data flow, skip conditions, and description associated with each instruction in the COP401L instruction set.

Table 2. COP401L Instruction Set Table Symbols

Symbol	Definition	Symbol	Definition
<b>INTERNAL ARCHITECTURE SYMBOLS</b>		<b>INSTRUCTION OPERAND SYMBOLS</b>	
A	4-bit Accumulator	d	4-bit Operand Field, 0-15 binary (RAM Digit Select)
B	6-bit RAM Address Register	r	2-bit Operand Field, 0-3 binary (RAM Register Select)
Br	Upper 2 bits of B (register address)	a	9-bit Operand Field, 0-511 binary (ROM Address)
Bd	Lower 4 bits of B (digit address)	y	4-bit Operand Field, 0-15 binary (Immediate Data)
C	1-bit Carry Register	RAM(s)	Contents of RAM location addressed by s
D	4-bit Data Output Port	ROM(t)	Contents of ROM location addressed by t
EN	4-bit Enable Register	<b>OPERATIONAL SYMBOLS</b>	
G	4-bit Register to latch data for G I/O Port	+	Plus
L	8-bit TRI-STATE <sup>®</sup> I/O Port	-	Minus
M	4-bit contents of RAM Memory pointed to by B Register	→	Replaces
PC	9-bit ROM Address Register (program counter)	↔	Is exchanged with
Q	8-bit Register to latch data for L I/O Port	=	Is equal to
SA	9-bit Subroutine Save Register A	$\bar{A}$	The one's complement of A
SB	9-bit Subroutine Save Register B	⊕	Exclusive-OR
SIO	4-bit Shift Register and Counter	:	Range of values
SK	Logic-Controlled Clock Output		

Table 3. COP401L Instruction Set

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>ARITHMETIC INSTRUCTIONS</b>						
ASC		30	<u>0 0 1 1</u> <u>0 0 0 0</u>	$A + C + \text{RAM}(B) \rightarrow A$ Carry $\rightarrow C$	Carry	Add with Carry, Skip on Carry
ADD		31	<u>0 0 1 1</u> <u>0 0 0 1</u>	$A + \text{RAM}(B) \rightarrow A$	None	Add RAM to A
AISC	y	5-	<u>0 1 0 1</u>   y	$A + y \rightarrow A$	Carry	Add Immediate, Skip on Carry (y $\neq$ 0)
CLRA		00	<u>0 0 0 0</u> <u>0 0 0 0</u>	$0 \rightarrow A$	None	Clear A
COMP		40	<u>0 1 0 0</u> <u>0 0 0 0</u>	$\bar{A} \rightarrow A$	None	One's complement of A to A
NOP		44	<u>0 1 0 0</u> <u>0 1 0 0</u>	None	None	No Operation
RC		32	<u>0 0 1 1</u> <u>0 0 1 0</u>	"0" $\rightarrow C$	None	Reset C
SC		22	<u>0 0 1 0</u> <u>0 0 1 0</u>	"1" $\rightarrow C$	None	Set C
XOR		02	<u>0 0 0 0</u> <u>0 0 1 0</u>	$A \oplus \text{RAM}(B) \rightarrow A$	None	Exclusive-OR RAM with A

Table 3. COP401L Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
TRANSFER OF CONTROL INSTRUCTIONS						
JID		FF	<u>1 1 1 1</u>   <u>1 1 1 1</u>	ROM (PC <sub>8,A,M</sub> ) → PC <sub>7:0</sub>	None	Jump Indirect (Note 2)
JMP	a	6--	<u>0 1 1 0</u>   <u>0 0 0</u>   <u>a<sub>7:0</sub></u>	a → PC	None	Jump
JP	a	--	<u>1</u>   a <sub>6:0</sub>	a → PC <sub>6:0</sub>	None	Jump within Page (Note 3)
			(pages 2,3 only) or <u>1 1</u>   a <sub>5:0</sub>	a → PC <sub>5:0</sub>		
JSRP	a	--	<u>1 0</u>   a <sub>5:0</sub>	PC + 1 → SA → SB 010 → PC <sub>8:6</sub> a → PC <sub>5:0</sub>	None	Jump to Subroutine Page (Note 4)
JSR	a	6--	<u>0 1 1 0</u>   <u>1 0 0</u>   <u>a<sub>7:0</sub></u>	PC + 1 → SA → SB a → PC	None	Jump to Subroutine
RET		48	<u>0 1 0 0</u>   <u>1 0 0 0</u>	SB → SA → PC	None	Return from Subroutine
RETSK		49	<u>0 1 0 0</u>   <u>1 0 0 1</u>	SB → SA → PC	Always Skip on Return	Return from Subroutine then Skip
MEMORY REFERENCE INSTRUCTIONS						
CAMQ		33	<u>0 0 1 1</u>   <u>0 0 1 1</u>	A → Q <sub>7:4</sub>	None	Copy A, RAM to Q
		3C	<u>0 0 1 1</u>   <u>1 1 0 0</u>	RAM(B) → Q <sub>3:0</sub>		
LD	r	-5	<u>0 0</u>  r  <u>0 1 0 1</u>	RAM(B) → A Br ⊕ r → Br	None	Load RAM into A, Exclusive-OR Br with r
LQID		BF	<u>1 0 1 1</u>   <u>1 1 1 1</u>	ROM(PC <sub>8,A,M</sub> ) → Q SA → SB	None	Load Q Indirect (Note 2)
RMB	0	4C	<u>0 1 0 0</u>   <u>1 1 0 0</u>	0 → RAM(B) <sub>0</sub>	None	Reset RAM Bit
	1	45	<u>0 1 0 0</u>   <u>0 1 0 1</u>	0 → RAM(B) <sub>1</sub>		
	2	42	<u>0 1 0 0</u>   <u>0 0 1 0</u>	0 → RAM(B) <sub>2</sub>		
	3	43	<u>0 1 0 0</u>   <u>0 0 1 1</u>	0 → RAM(B) <sub>3</sub>		
SMB	0	4D	<u>0 1 0 0</u>   <u>1 1 0 1</u>	1 → RAM(B) <sub>0</sub>	None	Set RAM Bit
	1	47	<u>0 1 0 0</u>   <u>0 1 1 1</u>	1 → RAM(B) <sub>1</sub>		
	2	46	<u>0 1 0 0</u>   <u>0 1 1 0</u>	1 → RAM(B) <sub>2</sub>		
	3	4B	<u>0 1 0 0</u>   <u>1 0 1 1</u>	1 → RAM(B) <sub>3</sub>		
STII	y	7-	<u>0 1 1 1</u>   y	y → RAM(B) Bd + 1 → Bd	None	Store Memory Immediate and Increment Bd
X	r	-6	<u>0 0</u>  r  <u>0 1 1 0</u>	RAM(B) ↔ A Br ⊕ r → Br	None	Exchange RAM with A, Exclusive-OR Br with r
XAD	3,15	23	<u>0 0 1 0</u>   <u>0 0 1 1</u>	RAM(3,15) ↔ A	None	Exchange A with RAM (3,15)
		BF	<u>1 0 1 1</u>   <u>1 1 1 1</u>			
XDS	r	-7	<u>0 0</u>  r  <u>0 1 1 1</u>	RAM(B) ↔ A Bd - 1 → Bd Br ⊕ r → Br	Bd decrements past 0	Exchange RAM with A and Decrement Bd, Exclusive-OR Br with r
XIS	r	-4	<u>0 0</u>  r  <u>0 1 0 0</u>	RAM(B) ↔ A Bd + 1 → Bd Br ⊕ r → Br	Bd increments past 15	Exchange RAM with A and Increment Bd, Exclusive-OR Br with r

Table 3. COP401L Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>REGISTER REFERENCE INSTRUCTIONS</b>						
CAB		50	<u>0 1 0 1</u>   <u>0 0 0 0</u>	A → Bd	None	Copy A to Bd
CBA		4E	<u>0 1 0 0</u>   <u>1 1 1 0</u>	Bd → A	None	Copy Bd to A
LBI	r,d	--	<u>0 0</u>  r (d-1) (d = 0, 9:15)	r,d → B	Skip until not a LBI	Load B Immediate with r,d (Note 5)
LEI	y	33 6-	<u>0 0 1 1</u>   <u>0 0 1 1</u> <u>0 1 1 0</u>  y	y → EN	None	Load EN Immediate (Note 6)
<b>TEST INSTRUCTIONS</b>						
SKC		20	<u>0 0 1 0</u>   <u>0 0 0 0</u>		C = "1"	Skip if C is True
SKE		21	<u>0 0 1 0</u>   <u>0 0 0 1</u>		A = RAM(B)	Skip if A Equals RAM
SKGZ		33 21	<u>0 0 1 1</u>   <u>0 0 1 1</u> <u>0 0 1 0</u>   <u>0 0 0 1</u>		G <sub>3:0</sub> = 0	Skip if G is Zero (all 4 bits)
SKGBZ		33	<u>0 0 1 1</u>   <u>0 0 1 1</u>	1st byte		Skip if G Bit is Zero
	0	01	<u>0 0 0 0</u>   <u>0 0 0 1</u>	} 2nd byte	G <sub>0</sub> = 0	
	1	11	<u>0 0 0 1</u>   <u>0 0 0 1</u>		G <sub>1</sub> = 0	
	2	03	<u>0 0 0 0</u>   <u>0 0 1 1</u>		G <sub>2</sub> = 0	
	3	13	<u>0 0 0 1</u>   <u>0 0 1 1</u>		G <sub>3</sub> = 0	
SKMBZ		01 11 03 13	<u>0 0 0 0</u>   <u>0 0 0 1</u> <u>0 0 0 1</u>   <u>0 0 0 1</u> <u>0 0 0 0</u>   <u>0 0 1 1</u> <u>0 0 0 1</u>   <u>0 0 1 1</u>		RAM(B) <sub>0</sub> = 0 RAM(B) <sub>1</sub> = 0 RAM(B) <sub>2</sub> = 0 RAM(B) <sub>3</sub> = 0	Skip if RAM Bit is Zero
<b>INPUT/OUTPUT INSTRUCTIONS</b>						
ING		22 2A	<u>1 0 0 1</u>   <u>1 0 0 1</u> <u>0 0 1 0</u>   <u>1 0 1 0</u>	C → A	None	Input C Ports to A
INL		33 2E	<u>0 0 1 1</u>   <u>0 0 1 1</u> <u>0 0 1 0</u>   <u>1 1 1 0</u>	L <sub>7:4</sub> → RAM(B) L <sub>3:0</sub> → A	None	Input L Ports to RAM, A
OBD		33 3E	<u>0 0 1 1</u>   <u>0 0 1 1</u> <u>0 0 1 1</u>   <u>1 1 1 0</u>	Bd → D	None	Output Bd to D Outputs
OMG		33 3A	<u>0 0 1 1</u>   <u>0 0 1 1</u> <u>0 0 1 1</u>   <u>1 0 1 0</u>	RAM(B) → G	None	Output RAM to G Ports
XAS		4F	<u>0 1 0 0</u>   <u>1 1 1 1</u>	A ↔ SIO, C → SKL	None	Exchange A with SIO (Note 2)

**Note 1:** All subscripts for alphabetical symbols indicate bit numbers unless explicitly defined (e.g., Br and Bd are explicitly defined). Bits are numbered 0 to N where 0 signifies the least significant bit (low-order, right-most bit). For example, A<sub>3</sub> indicates the most significant (left-most) bit of the 4-bit A register.

**Note 2:** For additional information on the operation of the XAS, JID, and LQID instructions, see below.

**Note 3:** The JP instruction allows a jump, while in subroutine pages 2 or 3, to any ROM location within the two-page boundary of pages 2 or 3. The JP instruction, otherwise, permits a jump to a ROM location within the current 64-word page. JP may not jump to the last word of a page.

**Note 4:** A JSRP transfers program control to subroutine page 2 (010 is loaded into the upper 3 bits of P). A JSRP may not be used when in pages 2 or 3. JSRP may not jump to the last word in page 2.

**Note 5:** The machine code for the lower 4 bits of the LBI instruction equals the binary value of the "d" data minus 1, e.g., to load the lower four bits of B (Bd) with the value 9 (1001<sub>2</sub>), the lower 4 bits of the LBI instruction equal 8 (1000<sub>2</sub>). To load 0, the lower 4 bits of the LBI instruction should equal 15 (1111<sub>2</sub>).

**Note 6:** Machine code for operand field y for LEI instruction should equal the binary value to be latched into EN, where a "1" or "0" in each bit of EN corresponds with the selection or deselection of a particular function associated with each bit. (See Functional Description, EN Register.)



The following information is provided to assist the user in understanding the operation of several unique instructions and to provide notes useful to programmers in writing COP401L programs.

### XAS Instruction

XAS (Exchange A with SIO) exchanges the 4-bit contents of the accumulator with the 4-bit contents of the SIO register. The contents of SIO will contain serial-in/serial-out shift register or binary counter data, depending on the value of the EN register. An XAS instruction will also affect the SK output. (See Functional Description, EN Register, above.) If SIO is selected as a shift register, an XAS instruction must be performed once every 4 instruction cycles to effect a continuous data stream.

### JID Instruction

JID (Jump Indirect) is an indirect addressing instruction, transferring program control to a new ROM location pointed to indirectly by A and M. It loads the lower 8 bits of the ROM address register PC with the contents of ROM addressed by the 9-bit word,  $PC_8, A, M$ .  $PC_8$  is not affected by this instruction.

Note that JID requires 2 instruction cycles to execute.

### LQID Instruction

LQID (Load Q Indirect) loads the 8-bit Q register with the contents of ROM pointed to by the 9-bit word  $PC_8, A, M$ . LQID can be used for table lookup or code conversion such as BCD to seven-segment. The LQID instruction "pushes" the stack ( $PC + 1 \rightarrow SA \rightarrow SB$ ) and replaces the least significant 8 bits of PC as follows:  $A \rightarrow PC_{7,4}$ ,  $RAM(B) \rightarrow PC_{3,0}$ , leaving  $PC_8$  unchanged. The ROM data pointed to by the new address is fetched and loaded into the Q latches. Next, the stack is "popped" ( $SB \rightarrow SA \rightarrow PC$ ), restoring the saved value of PC to continue sequential program execution. Since LQID pushes  $SA \rightarrow SB$ , the previous contents of SB are lost. Also, when LQID pops the stack, the previously pushed contents of SA are left in SB. The net result is that the contents of SA are placed in SB ( $SA \rightarrow SB$ ). Note that LQID takes two instruction cycle times to execute.

### Instruction Set Notes

- a. The first word of a COP401L program (ROM address 0) must be a CLRA (Clear A) instruction.
- b. Although skipped instructions are not executed, one instruction cycle time is devoted to skipping each byte of the skipped instruction. Thus all program paths except JID and LQID take the same number of cycle times whether instructions are skipped or executed. JID and LQID instructions take 2 cycles if executed and 1 cycle if skipped.
- c. The ROM is organized into 8 pages of 64 words each. The Program Counter is a 9-bit binary counter, and will count through page boundaries. If a JP, JSRP, JID or LQID instruction is located in the last word of a page, the instruction operates as if it were in the next page. For example: a JP located in the last word of a page will jump to a location in the next page. Also, a LQID or JID located in the last word of page 3 or 7 will access data in the next group of 4 pages.

## Typical Applications

### PROM-Based System

The COP401L may be used to emulate the COP410L. Figure 8 shows the interconnect to implement a COP401L hardware emulation. This connection uses one MM5204 EPROM as external memory. Other memory can be used such as bipolar PROM or RAM.

Pins  $IP_7-IP_0$  are bidirectional inputs and outputs. When the  $AD/\overline{DATA}$  clocking output turns on, the EPROM drivers are disabled and  $IP_7-IP_0$  output addresses. The 8-bit latch (MM74C373) latches the addresses to drive the memory.

When  $AD/\overline{DATA}$  turns off, the EPROM is enabled and the  $IP_7-IP_0$  pins will input the memory data. P8 outputs the most significant address bit to the memory. (SKIP output may be used for program debug if needed.)

24 of the COP401L pins may be configured exactly the same as a COP410L.

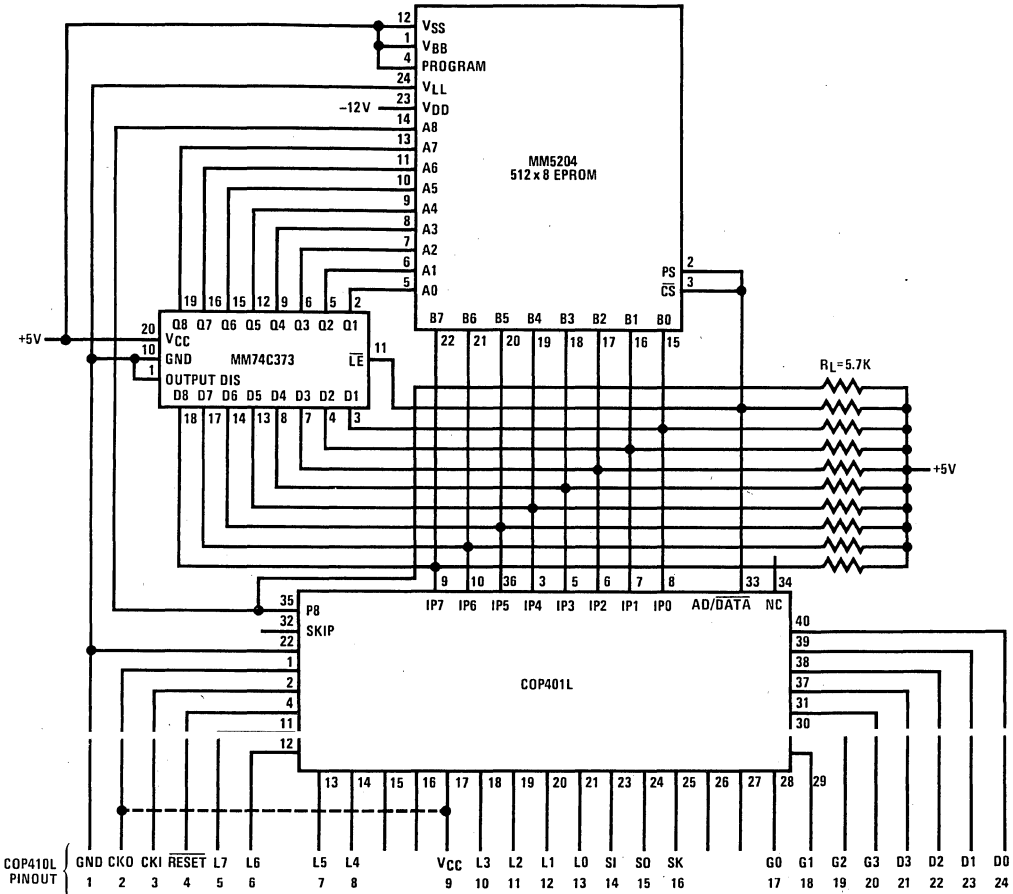


Figure 8. COP401L Used to Emulate a COP410L

### COP401L Mask Options

The following COP401L options have been implemented in this basic version of the COP401L.

Option Value	Comment	Option Value	Comment
Option 1 = 0	Ground — no option	Option 14 = 0	SI has load to $V_{CC}$
Option 2 = 1	CKO is RAM power supply input	Option 15 = 2	SO is push-pull output
Option 3 = N/A	CKI is external clock divide-by-32 (not available on COP410L)	Option 16 = 2	SK is push-pull output
Option 4 = 0	Reset has load to $V_{CC}$	Option 17 = 0	
Option 5 = 2		Option 18 = 0	G outputs are standard
Option 6 = 2		Option 19 = 0	
Option 7 = 2	L outputs are LED direct-drive	Option 20 = 0	
Option 8 = 2		Option 21 = 0	
Option 9 = 1	$V_{CC}$ pin 4.5V to 9.5V operation	Option 22 = 0	D outputs are standard very high current
Option 10 = 2		Option 23 = 0	
Option 11 = 2		Option 24 = 0	
Option 12 = 2	L outputs are LED direct-drive	Option 25 = 0	L
Option 13 = 2		Option 26 = 0	G Have standard TTL input levels
		Option 27 = 0	SI
		Option 28 = N/A	40-pin package

# COP402/COP402M and COP302/COP302M ROMless N-Channel Microcontrollers

## General Description

The COP402/COP402M and COP302/COP302M ROMless Microcontrollers are members of the COPS™ family, fabricated using N-channel silicon gate MOS technology. Each part contains CPU, RAM, and I/O, and is identical to a COP420 device, except the ROM has been removed; pins have been added to output the ROM address and to input ROM data. In a system, the COP402 or 402M will perform exactly as the COP420; this important benefit facilitates development and debug of a COP420 program prior to masking the final part. These devices are also appropriate in low volume applications, or when the program may require changing. The COP402M is identical to the COP402, except the MICROBUS™ interface option has been implemented.

The COP402 may also be used to emulate the COP410L, 411L, 420L or 420C by appropriately reducing the clock frequency. The COP302 and COP302M are the extended temperature range versions of the COP402 and COP402M.

## Features

- Low cost
- Exact circuit equivalent of COP420
- Standard 40-pin dual-in-line package
- Interfaces with standard PROM or ROM
- 64 x 4 RAM, addresses up to 1k x 8 ROM
- MICROBUS™ compatible (COP402M)
- Powerful instruction set
- True vectored interrupt, plus restart
- Three-level subroutine stack
- 4.0μs instruction time
- Single supply operation (4.5V to 6.3V)
- Internal time-base counter for real-time processing
- Internal binary counter register with MICROWIRE™ serial I/O capability
- Software/hardware compatible with other members of COP400 family
- Extended temperature range COP302 and COP302M (-40°C to +85°C) available

COPS, MICROBUS, and MICROWIRE are trademarks of National Semiconductor Corp.  
TRI-STATE is a registered trademark of National Semiconductor Corp.

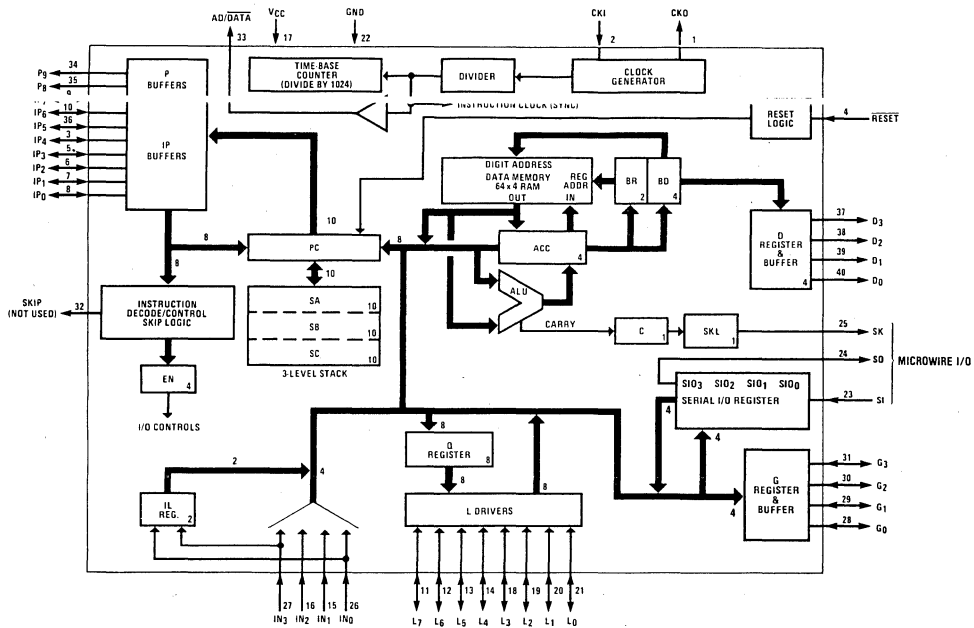


Figure 1. COP402/402M Block Diagram

**COP402/COP402M and COP302/COP302M****Absolute Maximum Ratings**

Voltage at Any Pin	-0.3V to +7V	Package Power Dissipation	750mW at 25°C 400mW at 70°C 250mW at 85°C
Operating Temperature Range	0°C to 70°C	Total Sink Current	50mA
COP402/COP402M	0°C to 70°C	Total Source Current	70mA
COP302/COP302M	-40°C to +85°C		
Storage Temperature Range	-65°C to +150°C		
Lead Temperature (soldering, 10 seconds)	300°C		

*Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

**COP402/COP402M****DC Electrical Characteristics**  $0^{\circ}\text{C} \leq T_A \leq 70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 6.3\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Operation Voltage		4.5	6.3	V
Power Supply Ripple	peak to peak (Note 3)		0.4	V
Supply Current	all outputs open $V_{CC} = 5\text{V}$		40	mA
Input Voltage Levels				
CKI Input Levels				
Crystal Input				
Logic High		2.4		V
Logic Low		-0.3	0.4	V
Schmitt Trigger Input				
RESET				
Logic High		$0.7 V_{CC}$		V
Logic Low		-0.3	0.6	V
All Other Inputs				
Logic High	$V_{CC} = \text{Max.}$	3.0		V
Logic High	$V_{CC} = 5\text{V} \pm 5\%$	2.0		V
Logic Low		-0.3	0.8	V
Input Load Source Current	$V_{CC} = 5\text{V}$ , $V_{IN} = 0\text{V}$	-100	-800	$\mu\text{A}$
Input Capacitance			7	pF
Hi-Z Input Leakage	$V_{CC} = 5\text{V}$	-1	+1	$\mu\text{A}$
Output Voltage levels				
D, G, L, SK, SO Outputs				
TTL Operation	$V_{CC} = 5\text{V} \pm 5\%$			
Logic High	$I_{OH} = -100\mu\text{A}$	2.4		V
Logic Low	$I_{OL} = 1.6\text{mA}$	-0.3	0.4	V
IP0-IP7, P8, P9, SKIP, CKO, AD/DATA				
Logic High	$I_{OH} = -75\mu\text{A}$	2.4		V
Logic Low	$I_{OL} = 400\mu\text{A}$	-0.3	0.4	V
CMOS Operation				
Logic High	$I_{OH} = -10\mu\text{A}$	$V_{CC} - 1$		V
Logic Low	$I_{OL} = 10\mu\text{A}$	-0.3	0.2	V
Output Current Levels				
LED Direct Drive (COP402)	$V_{CC} = 6\text{V}$			
Logic High	$V_{OH} = 2.0\text{V}$	2.5	14	mA
TRI-STATE® (COP402M) Leakage Current	$V_{CC} = 5\text{V}$	-2.5	+2.5	$\mu\text{A}$
Allowable Sink Current				
Per Pin (L, D, G)			10	mA
Per Pin (All Others)			2	mA
Per Port (L)			16	mA
Per Port (D, G)			10	mA
Allowable Source Current				
Per Pin (L)			-15	mA
Per Pin (All Others)			-1.5	mA

# COP302/COP302M

## DC Electrical Characteristics $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ , $4.5\text{V} \leq V_{CC} \leq 5.5\text{V}$ unless otherwise noted.

COP402/COP402M, COP302/COP302M

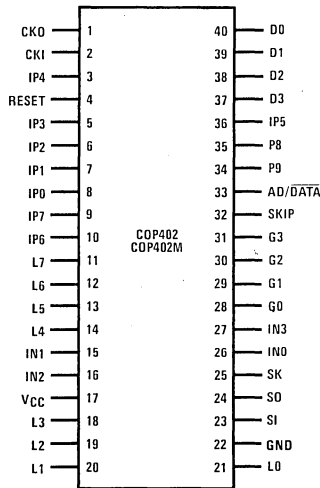
Parameter	Conditions	Min.	Max.	Units
Operation Voltage		4.5	5.5	V
Power Supply Ripple	peak to peak (Note 3)		0.4	V
Supply Current	$T_A = -40^{\circ}\text{C}$ , outputs open		50	mA
Input Voltage Levels				
CKI Input Levels				
Crystal Input				
Logic High		2.4		V
Logic Low		-0.3	0.3	V
Schmitt Trigger Input				
RESET				
Logic High		$0.7V_{CC}$		V
Logic Low		-0.3	0.4	V
All Other Inputs				
Logic High	$V_{CC} = \text{Max.}$	3.0		V
Logic High	$V_{CC} = 5\text{V} \pm 5\%$	2.2		V
Logic Low		-0.3	0.6	V
Input Load Source Current	$V_{CC} = 5\text{V}$ , $V_{IN} = 0\text{V}$	-100	-800	$\mu\text{A}$
Input Capacitance			7	pF
Hi-Z Input Leakage	$V_{CC} = 5\text{V}$	-2	+2	$\mu\text{A}$
Output Voltage Levels				
D, G, L, SK, SO Outputs				
TTL Operation	$V_{CC} = 5\text{V} \pm 5\%$			
Logic High	$I_{OH} = -75\mu\text{A}$	2.4		V
Logic Low	$I_{OL} = 1.6\text{mA}$	-0.3	0.4	V
IP0-IP7, P8, P9, SKIP, CKO, AD/DATA				
Logic High	$I_{OH} = -75\mu\text{A}$	2.4		V
Logic Low	$I_{OL} = 400\mu\text{A}$	-0.3	0.4	V
CMOS Operation				
Logic High	$I_{OH} = -10\mu\text{A}$	$V_{CC} - 1$		V
Logic Low	$I_{OL} = 10\mu\text{A}$	-0.3	0.2	V
Output Current Levels				
LED Direct Drive (COP302)	$V_{CC} = 5\text{V}$ (Note 4)			
Logic High	$V_{OH} = 2.0\text{V}$	1.0	12	mA
CKI Sink Current (R/C Option)	$V_{IN} = 3.5\text{V}$	2		mA
CKO (RAM Supply Current)	$V_R = 3.3\text{V}$		4	mA
TRI-STATE® (COP302M) Leakage Current	$V_{CC} = 5\text{V}$	-5	+5	$\mu\text{A}$
Allowable Sink Current				
Per Pin (L, D, G)			10	mA
Per Pin (All Others)			2	mA
Per Port (L)			16	mA
Per Port (D, G)			10	mA
Allowable Source Current				
Per Pin (L)			-15	mA
Per Pin (All Others)			-1.5	mA



**AC Electrical Characteristics**COP402/COP402M  $0^{\circ}\text{C} \leq T_A \leq 70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{\text{CC}} \leq 6.3\text{V}$  unless otherwise noted.COP302/COP302M  $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{\text{CC}} \leq 5.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Instruction Cycle Time		4	10	$\mu\text{s}$
Operating CKI Frequency	$\div 16$ mode	1.6	4.0	MHz
CKI Duty Cycle (Note 1)		40	60	%
Rise Time	Freq. = 4 MHz		60	ns
Fall Time	Freq. = 4 MHz		40	ns
Inputs:				
SI				
$t_{\text{SETUP}}$		0.3		$\mu\text{s}$
$t_{\text{HOLD}}$		250		ns
All Other Inputs				
$t_{\text{SETUP}}$		1.7		$\mu\text{s}$
$t_{\text{HOLD}}$		300		ns
Output Propagation Delay	Test Conditions: $R_L = 5\text{k}$ , $C_L = 50\text{pF}$ , $V_{\text{OUT}} = 1.5\text{V}$			
SO and SK				
$t_{\text{pd1}}$			1.0	$\mu\text{s}$
$t_{\text{pd0}}$			1.0	$\mu\text{s}$
CKO				
$t_{\text{pd1}}$			0.25	$\mu\text{s}$
$t_{\text{pd0}}$			0.25	$\mu\text{s}$
AD/DATA, SKIP				
$t_{\text{pd1}}$			0.6	$\mu\text{s}$
$t_{\text{pd0}}$			0.6	$\mu\text{s}$
All Other Outputs				
$t_{\text{pd1}}$			1.4	$\mu\text{s}$
$t_{\text{pd0}}$			1.4	$\mu\text{s}$
MICROBUS™ Timing	$C_L = 100\text{pF}$ , $V_{\text{CC}} = 5\text{V} \pm 5\%$			
Read Operation (Figure 4)				
Chip Select Stable before $\overline{\text{RD}}$ — $t_{\text{CSR}}$		65		ns
Chip Select Hold Time for $\overline{\text{RD}}$ — $t_{\text{RCS}}$		20		ns
$\overline{\text{RD}}$ Pulse Width— $t_{\text{RR}}$		400		ns
Data Delay from $\overline{\text{RD}}$ — $t_{\text{RD}}$			375	ns
$\overline{\text{RD}}$ to Data Floating— $t_{\text{DF}}$			250	ns
Write Operation (Figure 5)				
Chip Select Stable before $\overline{\text{WR}}$ — $t_{\text{CSW}}$		65		ns
Chip Select Hold Time for $\overline{\text{WR}}$ — $t_{\text{WCS}}$		20		ns
$\overline{\text{WR}}$ Pulse Width— $t_{\text{WW}}$		400		ns
Data Set-Up Time for $\overline{\text{WR}}$ — $t_{\text{DW}}$		320		ns
Data Hold Time for $\overline{\text{WR}}$ — $t_{\text{WD}}$		100		ns
INTR Transition Time from $\overline{\text{WR}}$ — $t_{\text{WI}}$			700	ns

**Note 1:** Duty cycle =  $t_{\text{W1}} / (t_{\text{W1}} + t_{\text{W0}})$ .**Note 2:** See Figure 9 for additional I/O characteristics.**Note 3:** Voltage change must be less than 0.5 volts in a 1ms period.**Note 4:** Exercise great care not to exceed maximum device power dissipation limits when direct driving LEDs (or sourcing similar loads) at high temperature.



Order Number COP402N, COP402MN  
NS Package N40A

Figure 2. Connection Diagram

Pin	Description	Pin	Description
L7-L0	8 bidirectional I/O ports with TRI-STATE®	AD/DATA	Address out/data in flag
G3-G0	4 bidirectional I/O ports	SKIP	Instruction skip output
D3-D0	4 general purpose outputs	CKI	System oscillator input
IN3-IN0	4 general purpose inputs	CKO	System oscillator output
SI	Serial input (or counter input)	RESET	System reset input
SO	Serial output (or general purpose output)	VCC	Power supply
SK	Logic-controlled clock (or general purpose output)	GND	Ground
		IP7-IP0	8 bidirectional ROM address and data ports
		P8, P9	2 most significant ROM address outputs

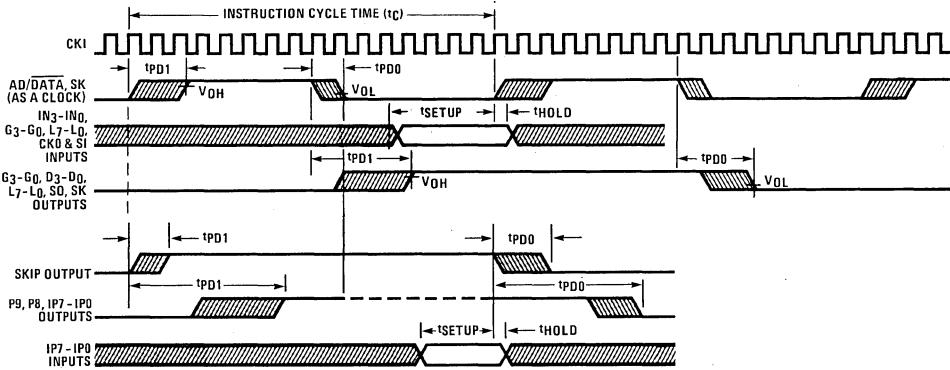


Figure 3a. Input/Output Timing Diagrams (Crystal + 16 Mode)

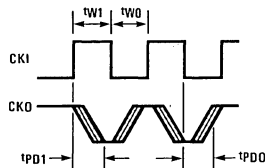


Figure 3b. CKO Output Timing



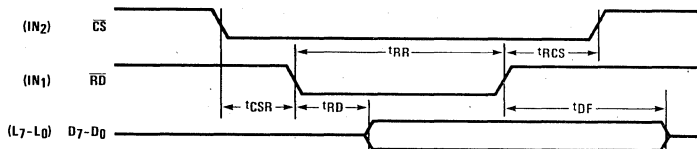


Figure 4. MICROBUSTM Read Operation Timing

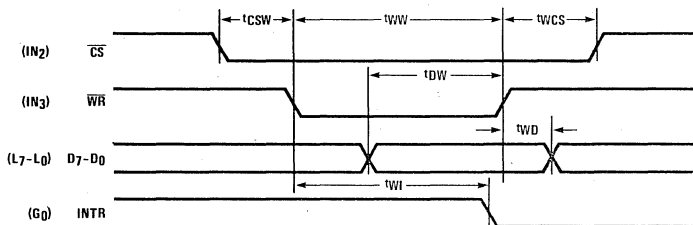


Figure 5. MICROBUSTM Write Operation Timing

## Functional Description

A block diagram of the COP402 is given in Figure 1. Data paths are illustrated in simplified form to depict how the various logic elements communicate with each other in implementing the instruction set of the device. Positive logic is used. When a bit is set, it is a logic "1" (greater than 2 volts). When a bit is reset, it is a logic "0" (less than 0.8 volts).

### Program Memory

Program Memory consists of a 1,024-byte external memory (typically PROM). Words of this memory may be program instructions, program data or ROM addressing data. Because of the special characteristics associated with the JP, JSRP, JID and LQID instructions, ROM must often be thought of as being organized into 16 pages of 64 words each.

ROM addressing is accomplished by a 10-bit PC register. Its binary value selects one of the 1,024 8-bit words contained in ROM. A new address is loaded into the PC register during each instruction cycle. Unless the instruction is a transfer of control instruction, the PC register is loaded with the next sequential 10-bit binary count value. Three levels of subroutine nesting are implemented by the 10-bit subroutine save registers, SA, SB and SC, providing a last-in, first-out (LIFO) hardware subroutine stack.

ROM instruction words are fetched, decoded and executed by the Instruction Decode, Control and Skip Logic circuitry.

### Data Memory

Data memory consists of a 256-bit RAM, organized as 4 data registers of 16 4-bit digits. RAM addressing is implemented by a 6-bit B register whose upper 2 bits (Br) select 1 of 4 data registers and lower 4 bits (Bd) select 1 of 16 4-bit digits in the selected data register. While the 4-bit contents of the selected RAM digit (M) is usually loaded into or from, or exchanged with, the A register (accumulator), it may also be loaded into

or from the Q latches or loaded from the L ports. RAM addressing may also be performed directly by the LDD and XAD instructions based upon the 6-bit contents of the operand field of these instructions. The Bd register also serves as a source register for 4-bit data sent directly to the D outputs.

### Internal Logic

The 4-bit **A register** (accumulator) is the source and destination register for most I/O, arithmetic, logic and data memory access operations. It can also be used to load the Br and Bd portions of the B register, to load and input 4 bits of the 8-bit Q latch data, to input 4 bits of the 8-bit L I/O port data and to perform data exchanges with the SIO register.

A 4-bit **adder** performs the arithmetic and logic functions of the COP402/402M, storing its results in A. It also outputs a carry bit to the 1-bit **C register**, most often employed to indicate arithmetic overflow. The C register, in conjunction with the XAS instruction and the EN register, also serves to control the SK output. C can be outputted directly to SK or can enable SK to be a sync clock each instruction cycle time. (See XAS instruction and EN register description, below.)

Four **general-purpose inputs**,  $IN_3$ - $IN_0$ , are provided;  $IN_1$ ,  $IN_2$  and  $IN_3$  may be selected, by a mask-programmable option, as Read Strobe, Chip Select and Write Strobe inputs, respectively, for use in MICROBUSTM applications.

The **D register** provides 4 general-purpose outputs and is used as the destination register for the 4-bit contents of Bd.

The **G register** contents are outputs to 4 general-purpose bidirectional I/O ports.  $G_0$  may be mask-programmed as a "ready" output for MICROBUSTM applications.

The **Q register** is an internal, latched, 8-bit register, used to hold data loaded to or from M and A, as well as 8-bit data from ROM. Its contents are output to the L I/O ports when the L drivers are enabled under program control. (See LEI instruction.) With the MICROBUS™ option selected, Q can also be loaded with the 8-bit contents of the L I/O ports upon the occurrence of a write strobe from the host CPU.

The **8 L drivers**, when enabled, output the contents of latched Q data to the L I/O ports. Also, the contents of L may be read directly into A and M. As explained above, the MICROBUS™ option allows L I/O port data to be latched into the Q register. L I/O ports can be directly connected to the segments of a multiplexed LED display (using the LED Direct Drive output configuration option) with Q data being outputted to the Sa–Sg and decimal point segments of the display.

The **SIO register** functions as a 4-bit serial-in/serial-out shift register or as a binary counter depending on the contents of the EN register. (See EN register description, below.) Its contents can be exchanged with A, allowing it to input or output a continuous serial data stream. SIO may also be used to provide additional parallel I/O by connecting SO to external serial-in/parallel-out shift registers.

The **XAS instruction** copies C into the SKL latch. In the counter mode, SK is the output of SKL. In the shift register mode, SK outputs SKL ANDed with internal instruction cycle clock.

The **EN register** is an internal 4-bit register loaded under program control by the LEI instruction. The state of each bit of this register selects or deselects the particular feature associated with each bit of the EN register (EN<sub>3</sub>–EN<sub>0</sub>).

1. The least significant bit of the enable register, EN<sub>0</sub>, selects the SIO register as either a 4-bit shift register or a 4-bit binary counter. With EN<sub>0</sub> set, SIO is an asynchronous binary counter, *decrementing* its value by one upon each low-going pulse ("1" to "0") occurring on the SI input. Each pulse must be at least two instruction cycles wide. SK outputs the value of SKL. The SO output is equal to the value of EN<sub>3</sub>. With EN<sub>0</sub> reset, SIO is a serial shift

Register shifting left each instruction cycle time. The data present at SI goes into the least significant bit of SIO. SO can be enabled to output the most significant bit of SIO each cycle time. (See 4 below.) The SK output becomes a logic-controlled clock.

2. With EN<sub>1</sub> set the IN<sub>1</sub> input is enabled as an interrupt input. Immediately following an interrupt, EN<sub>1</sub> is reset to disable further interrupts.
3. With EN<sub>2</sub> set, the L drivers are enabled to output the data in Q to the L I/O ports. Resetting EN<sub>2</sub> disables the L drivers, placing the L I/O ports in a high-impedance input state. If the MICROBUS™ option is being used, EN<sub>2</sub> does not affect the L drivers.
4. EN<sub>3</sub>, in conjunction with EN<sub>0</sub>, affects the SO output. With EN<sub>0</sub> set (binary counter option selected) SO will output the value loaded into EN<sub>3</sub>. With EN<sub>0</sub> reset (serial shift register option selected), setting EN<sub>3</sub> enables SO as the output of the SIO shift register, outputting serial shifted data each instruction time. Resetting EN<sub>3</sub> with the serial shift register option selected disables SO as the shift register output; data continues to be shifted through SIO and can be exchanged with A via an XAS instruction but SO remains reset to "0." The table below provides a summary of the modes associated with EN<sub>3</sub> and EN<sub>0</sub>.

### Interrupt

The following features are associated with the IN<sub>1</sub> interrupt procedure and protocol and must be considered by the programmer when utilizing interrupts.

- a. The interrupt, once acknowledged as explained below, pushes the next sequential program counter address (PC + 1) onto the stack, pushing in turn the contents of the other subroutine-save registers to the next lower level (PC + 1 → SA → SB → SC). Any previous contents of SC are lost. The program counter is set to hex address 0FF (the last word of page 3) and EN<sub>1</sub> is reset.

Enable Register Modes — Bits EN<sub>3</sub> and EN<sub>0</sub>

EN <sub>3</sub>	EN <sub>0</sub>	SIO	SI	SO	SK
0	0	Shift Register	Input to Shift Register	0	If SKL = 1, SK = SYNC If SKL = 0, SK = 0
1	0	Shift Register	Input to Shift Register	Serial Out	If SKL = 1, SK = SYNC If SKL = 0, SK = 0
0	1	Binary Counter	Input to Binary Counter	0	If SKL = 1, SK = 1 If SKL = 0, SK = 0
1	1	Binary Counter	Input to Binary Counter	1	If SKL = 1, SK = 1 If SKL = 0, SK = 0

- b. An interrupt will be acknowledged only after the following conditions are met:
1.  $EN_1$  has been set.
  2. A low-going pulse ("1" to "0") at least two instruction cycles wide occurs on the  $IN_1$  input.
  3. A currently executing instruction has been completed.
  4. All successive transfer of control instructions and successive LBIs have been completed (e.g., if the main program is executing a JP instruction which transfers program control to another JP instruction, the interrupt will not be acknowledged until the second JP instruction has been executed).
- c. Upon acknowledgement of an interrupt, the skip logic status is saved and later restored upon the popping of the stack. For example, if an interrupt occurs during the execution of ASC (Add with Carry, Skip on Carry) instruction which results in carry, the skip logic status is saved and program control is transferred to the interrupt servicing routine at hex address OFF. At the end of the interrupt routine, a RET instruction is executed to "pop" the stack and return program control to the instruction following the original ASC. At this time, the skip logic is enabled and skips this instruction because of the previous ASC carry. Subroutines and the LQID instruction should not be nested within the interrupt servicing routine since their popping of the stack enables any previously saved main program skips, interfering with the orderly execution of the interrupt routine.
- d. The first instruction of the interrupt routine at hex address OFF must be a NOP.
- e. A LEI instruction can be put immediately before the RET to re-enable interrupts.

### MICROBUS™ Interface

The COP402M can be used as a peripheral microprocessor device, inputting and outputting data from and to a host microprocessor ( $\mu P$ ).  $IN_1$ ,  $IN_2$ , and  $IN_3$  general purpose inputs become MICROBUS™ compatible read-strobe, chip-select, and write-strobe lines, respectively.  $IN_1$  becomes  $\overline{RD}$  — a logic "0" on this input will cause Q latch data to be enabled to the L ports for input to the  $\mu P$ .  $IN_2$  becomes  $\overline{CS}$  — a logic "0" on this line selects the COP402M as the  $\mu P$  peripheral device by enabling the operation of the  $\overline{RD}$  and  $\overline{WR}$  lines and allows for the selection of one of several peripheral components.  $IN_3$  becomes  $\overline{WR}$  — a logic "0" on this line will write bus data from the L ports to the Q latches for input to the COP402M.  $G_0$  becomes INTR, a "ready" output reset by a write pulse from the  $\mu P$  on the  $\overline{WR}$  line, providing the "handshaking" capability necessary for asynchronous data transfer between the host CPU and the COP402M.

This option has been designed for compatibility with National's MICROBUS™ — a standard interconnect

system for 8-bit parallel data transfer between MOS/LSI CPUs and interfacing devices. (See MICROBUS™, National Publication.) The functioning and timing relationships between the COP402M signal lines affected by this option are as specified for the MICROBUS™ interface, and are given in the AC electrical characteristics and shown in the timing diagrams (Figures 4 and 5). Connection to the MICROBUS™ is shown in Figure 6.

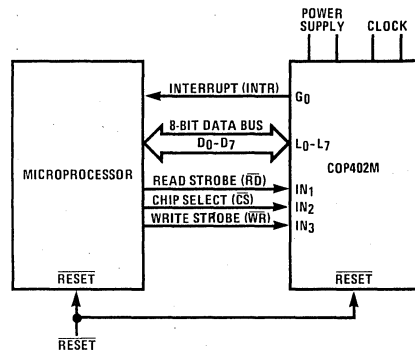


Figure 6. MICROBUS™ Option Interconnect

### Initialization

The Reset Logic will initialize (clear) the device upon power-up if the power supply rise time is less than 1ms and greater than  $1\mu s$ . If the power supply rise time is greater than 1ms, the user must provide an external RC network and diode to the  $\overline{RESET}$  pin as shown below. The  $\overline{RESET}$  pin is configured as a Schmitt trigger input. If not used it should be connected to  $V_{CC}$ . Initialization will occur whenever a logic "0" is applied to the  $\overline{RESET}$  input, provided it stays low for at least two instruction cycle times.

Upon initialization, the PC register is cleared to 0 (ROM address 0) and the A, B, C, D, EN, G, and SO are cleared. The SK output is enabled as a SYNC output, providing a pulse each instruction cycle time. *Data Memory (RAM) is not cleared upon initialization.* The first instruction at address 0 must be a CLRA.

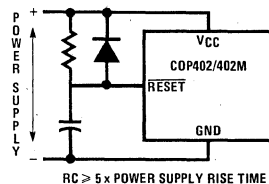
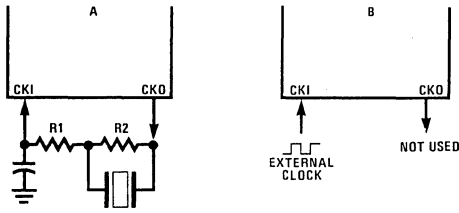


Figure 7. Power-Up Clear Circuit

**Oscillator**

There are two basic clock oscillator configurations available as shown by Figure 8.

- a. **Crystal Controlled Oscillator.** CKI and CKO are connected to an external crystal. The instruction cycle time equals the crystal frequency divided by 16.
- b. **External Oscillator.** CKI is driven by an external clock signal. The instruction cycle time is the clock frequency divided by 16.



Crystal Value	Component Values		
	R1	R2	C
4MHz	1k	1M	27pF
3.58MHz	1k	1M	27pF
2.09MHz	1k	1M	56pF

Figure 8. COP402/402M Oscillator

**External Memory Interface**

The COP402 and COP402M are designed for use with an external Program memory. This memory may be implemented using any devices having the following characteristics:

1. random addressing
2. TTL-compatible TRI-STATE® outputs
3. TTL-compatible inputs
4. access time = 1.0µs, max.

Typically these requirements are met using bipolar or MOS PROMs.

During operation, the address of the next instruction is sent out on P9, P8, and IP7 through IP0 during the time that AD/DATA is high (logic "1" = address mode). Address data on the IP lines is stored into an external latch on the high-to-low transition of the AD/DATA line; P9 and P8 are dedicated address outputs, and do not need to be latched. When AD/DATA is low (logic "0" = data mode), the output of the memory is gated onto IP7 through IP0, forming the input bus. Note that the AD/DATA output has a period of one instruction time, a duty cycle of approximately 50%, and specifies whether the IP lines are used for address output or instruction input. A simplified block diagram of the external memory interface is shown in Figure 9.

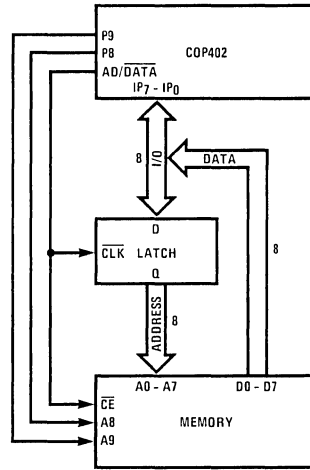


Figure 9. External Memory Interface to COP402

**Input/Output**

COP402 outputs have the following configurations, illustrated in Figure 9:

- a. **Standard** — an enhancement-mode device to ground in conjunction with a depletion-mode device to V<sub>CC</sub>, compatible with TTL and CMOS input requirements.
- b. **High Drive** — same as a. except greater current sourcing capability.
- c. **Push-Pull** — an enhancement-mode device to ground in conjunction with a depletion-mode device paralleled by an enhancement-mode device to V<sub>CC</sub>. This configuration has been provided to allow for fast rise and fall times when driving capacitive loads.
- d. **LED Direct Drive** — an enhancement-mode device to ground and to V<sub>CC</sub>, meeting the typical current sourcing requirements of the segments of an LED display. The sourcing device is clamped to limit current flow. These devices may be turned off under program control (see Functional Description, EN Register), placing the outputs in a high-impedance state to provide required LED segment blanking for a multiplexed display.
- e. **TRI-STATE® Push-Pull** — an enhancement-mode device to ground and V<sub>CC</sub> intended to meet the requirements associated with the MICROBUS™ option. These outputs are TRI-STATE® outputs, allowing for connection of these outputs to a data bus shared by other bus drivers.
- f. Inputs have an on-chip depletion load device to V<sub>CC</sub>, as shown in Figure 10f.

The above input and output configurations share common enhancement-mode and depletion-mode devices. Specifically, all configurations use one or

more of six devices (numbered 1-6, respectively). Minimum and maximum current ( $I_{OUT}$  and  $V_{OUT}$ ) curves are given in Figure 10 for each of these devices.

The SO,SK outputs are configured as shown in Figure 10c. The D and G outputs are configured as shown in Figure 10a. Note that when inputting data to the G ports, the G outputs should be set to "1." The L outputs are configured as in Figure 10d on the COP402. On the COP402M the L outputs are as in figure 10e.

An important point to remember if using configuration d with the L drivers is that even when the L drivers are disabled, the depletion load device will source a small amount of current. (See Figure 11.)

IP7 through IP0 outputs are configured as shown in Figure 10c; P9, P8, SKIP, and AD/DATA are configured as shown in Figure 10b.

**COP402/402M Instruction Set**

Table 1 is a symbol table providing internal architecture, instruction operand and operational symbols used in the instruction set table.

Table 2 provides the mnemonic, operand, machine code, data flow, skip conditions and description associated with each instruction in the COP402/402M instruction set.

The following information is provided to assist the user in understanding the operation of several unique instructions and to provide notes useful to programmers in writing programs.

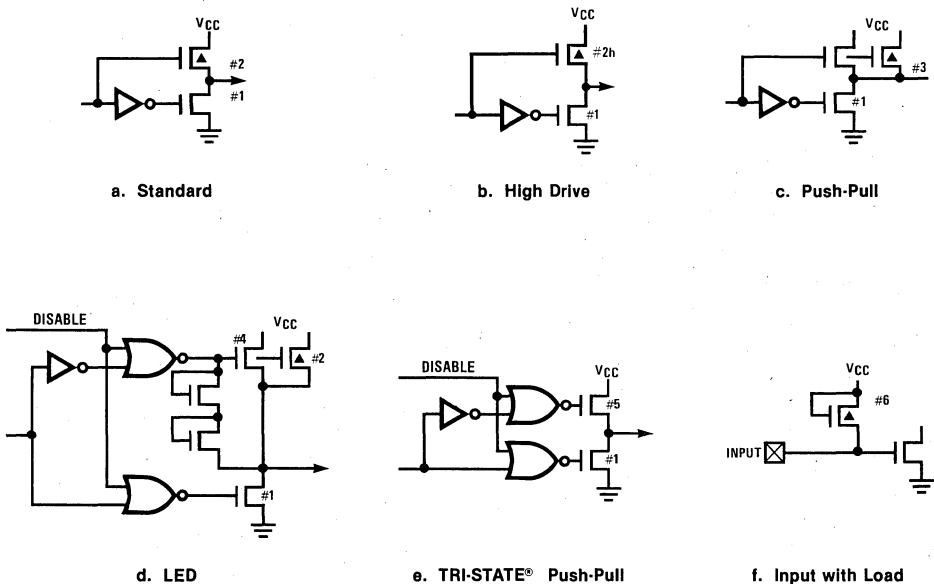
**XAS Instruction**

XAS (Exchange A with SIO) exchanges the 4-bit contents of the accumulator with the 4-bit contents of the SIO register. The contents of SIO will contain serial-in/serial-out shift register or binary counter data, depending on the value of the EN register. An XAS instruction will also affect the SK output. (See Functional Description, EN Register, above.) If SIO is selected as a shift register, an XAS instruction must be performed once every 4 instruction cycles to effect a continuous data stream.

**JID Instruction**

JID (Jump Indirect) is an indirect addressing instruction, transferring program control to a new ROM location pointed to indirectly by A and M. It loads the lower 8 bits of the ROM address register PC with the contents of ROM addressed by the 10-bit word,  $PC_{9:8}$ , A, M.  $PC_9$  and  $PC_8$  are not affected by this instruction.

Note that JID requires 2 instruction cycles.



(▲ IS DEPLETION DEVICE)

Figure 10. Input/Output Configurations

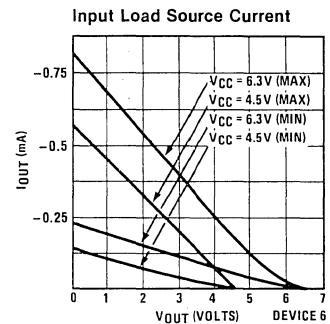
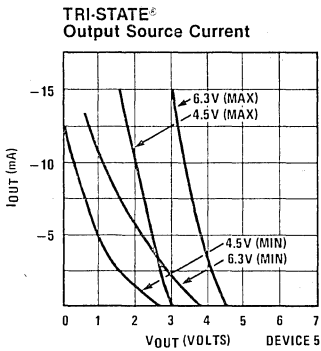
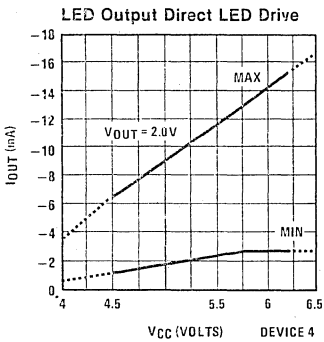
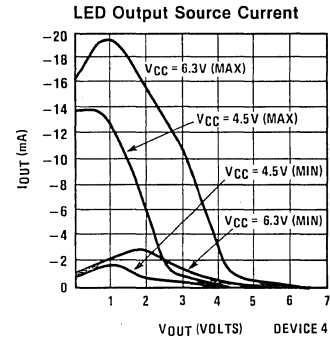
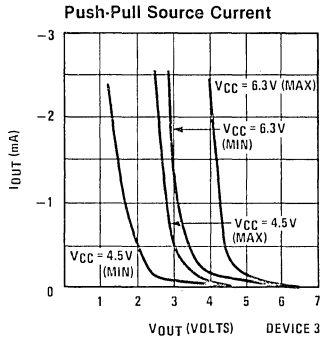
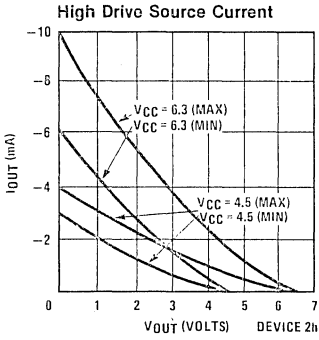
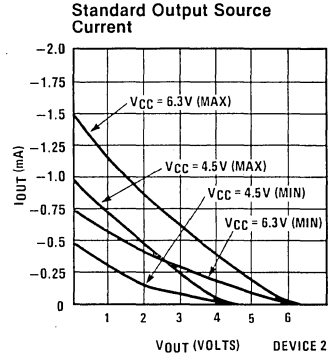
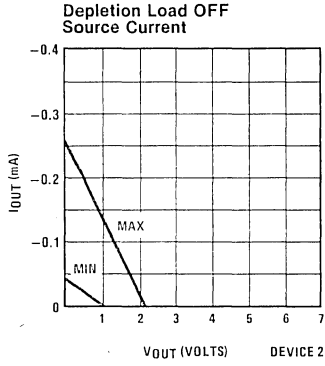
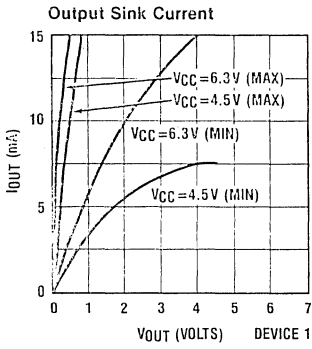


Figure 11. COP402/COP402M Input/Output Characteristics

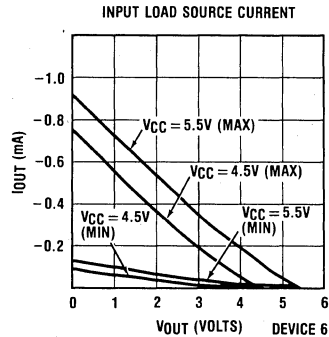
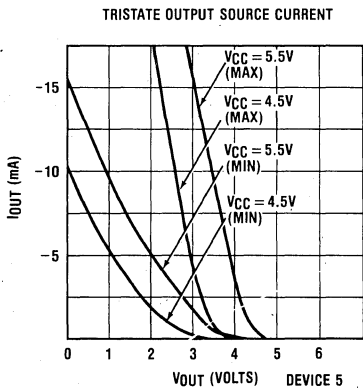
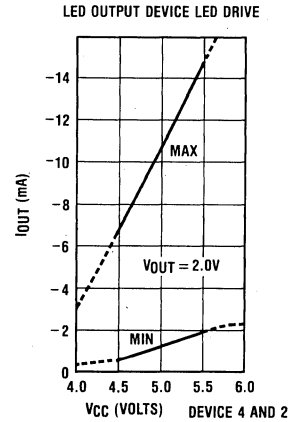
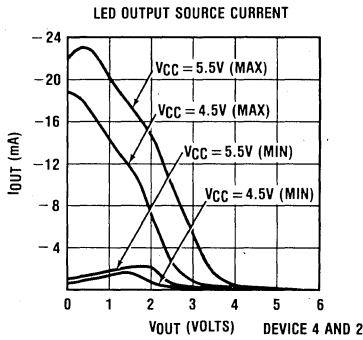
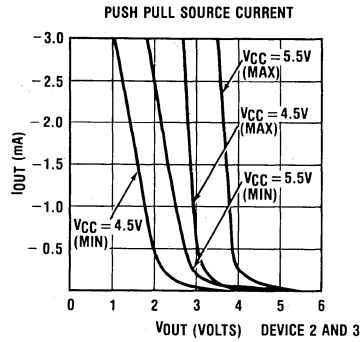
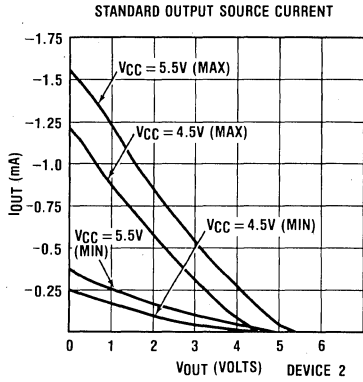
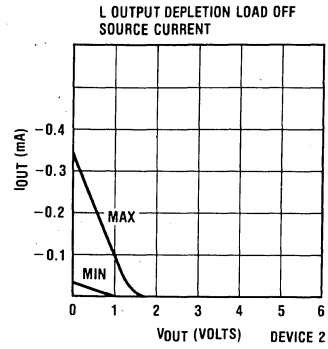
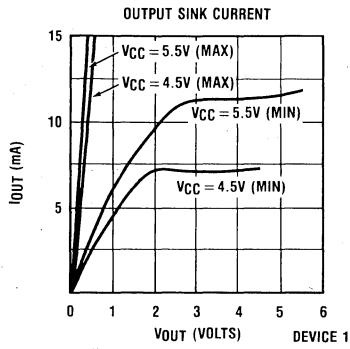


Figure 11a. COP302/COP302M Input/Output Characteristics





Table 2. COP402/COP402M Instruction Set Table (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description				
<b>TRANSFER OF CONTROL INSTRUCTIONS</b>										
JID		FF	<table border="1"><tr><td>1111</td><td>1111</td></tr></table>	1111	1111	ROM (PC <sub>9:8</sub> , A, M) → PC <sub>7:0</sub>	None	Jump Indirect (Note 3)		
1111	1111									
JMP	a	6-	<table border="1"><tr><td>0110</td><td>00a<sub>9:8</sub></td></tr><tr><td colspan="2">a<sub>7:0</sub></td></tr></table>	0110	00a <sub>9:8</sub>	a <sub>7:0</sub>		a → PC	None	Jump
0110	00a <sub>9:8</sub>									
a <sub>7:0</sub>										
JP	a	--	<table border="1"><tr><td>1</td><td>a<sub>6:0</sub></td></tr></table> (pages 2,3 only)	1	a <sub>6:0</sub>	a → PC <sub>6:0</sub>	None	Jump within Page (Note 4)		
			1	a <sub>6:0</sub>						
or <table border="1"><tr><td>11</td><td>a<sub>5:0</sub></td></tr></table> (all other pages)	11	a <sub>5:0</sub>	a → PC <sub>5:0</sub>							
11	a <sub>5:0</sub>									
JSRP	a	--	<table border="1"><tr><td>10</td><td>a<sub>5:0</sub></td></tr></table>	10	a <sub>5:0</sub>	PC + 1 → SA → SB → SC 0010 → PC <sub>9:6</sub> a → PC <sub>5:0</sub>	None	Jump to Subroutine Page (Note 5)		
10	a <sub>5:0</sub>									
JSR	a	6-	<table border="1"><tr><td>0110</td><td>10a<sub>9:8</sub></td></tr></table>	0110	10a <sub>9:8</sub>	PC + 1 → SA → SB → SC	None	Jump to Subroutine		
			0110	10a <sub>9:8</sub>						
<table border="1"><tr><td colspan="2">a<sub>7:0</sub></td></tr></table>	a <sub>7:0</sub>		a → PC							
a <sub>7:0</sub>										
RET		48	<table border="1"><tr><td>0100</td><td>1000</td></tr></table>	0100	1000	SC → SB → SA → PC	None	Return from Subroutine		
0100	1000									
RETSK		49	<table border="1"><tr><td>0100</td><td>1001</td></tr></table>	0100	1001	SC → SB → SA → PC	Always Skip on Return	Return from Subroutine then Skip		
0100	1001									
<b>MEMORY REFERENCE INSTRUCTIONS</b>										
CAMQ		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	A → Q <sub>7:4</sub>	None	Copy A, RAM to Q		
		0011	0011							
3C	<table border="1"><tr><td>0011</td><td>1100</td></tr></table>	0011	1100	RAM(B) → Q <sub>3:0</sub>						
0011	1100									
CQMA		33	<table border="1"><tr><td>0011</td><td>0011</td></tr></table>	0011	0011	Q <sub>7:4</sub> → RAM(B)	None	Copy Q to RAM, A		
		0011	0011							
2C	<table border="1"><tr><td>0010</td><td>1100</td></tr></table>	0010	1100	Q <sub>3:0</sub> → A						
0010	1100									
LD	r	-5	<table border="1"><tr><td>00</td><td>r</td><td>0101</td></tr></table>	00	r	0101	RAM(B) → A Br ⊕ r → Br	None	Load RAM into A, Exclusive-OR Br with r	
00	r	0101								
LDD	r,d	23	<table border="1"><tr><td>0010</td><td>0011</td></tr></table>	0010	0011	RAM(r,d) → A	None	Load A with RAM pointed to directly by r,d		
			0010	0011						
<table border="1"><tr><td>00</td><td>r</td><td>d</td></tr></table>	00	r	d							
00	r	d								
LQID		BF	<table border="1"><tr><td>1011</td><td>1111</td></tr></table>	1011	1111	ROM(PC <sub>9:8</sub> , A, M) → Q SB → SC	None	Load Q Indirect (Note 3)		
1011	1111									
RMB		0	4C	<table border="1"><tr><td>0100</td><td>1100</td></tr></table>	0100	1100	0 → RAM(B) <sub>0</sub>	None	Reset RAM Bit	
		0100	1100							
		1	45	<table border="1"><tr><td>0100</td><td>0101</td></tr></table>	0100	0101	0 → RAM(B) <sub>1</sub>			
		0100	0101							
2	42	<table border="1"><tr><td>0100</td><td>0010</td></tr></table>	0100	0010	0 → RAM(B) <sub>2</sub>					
0100	0010									
3	43	<table border="1"><tr><td>0100</td><td>0011</td></tr></table>	0100	0011	0 → RAM(B) <sub>3</sub>					
0100	0011									
SMB		0	4D	<table border="1"><tr><td>0100</td><td>1101</td></tr></table>	0100	1101	1 → RAM(B) <sub>0</sub>	None	Set RAM Bit	
		0100	1101							
		1	47	<table border="1"><tr><td>0100</td><td>1101</td></tr></table>	0100	1101	1 → RAM(B) <sub>1</sub>			
		0100	1101							
2	46	<table border="1"><tr><td>0100</td><td>0110</td></tr></table>	0100	0110	1 → RAM(B) <sub>2</sub>					
0100	0110									
3	4B	<table border="1"><tr><td>0100</td><td>1011</td></tr></table>	0100	1011	1 → RAM(B) <sub>3</sub>					
0100	1011									

Table 2. COP402/COP402M Instruction Set Table (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>MEMORY REFERENCE INSTRUCTIONS (continued)</b>						
STII	y	7-	0111 y	y → RAM(B) Bd + 1 → Bd	None	Store Memory Immediate and Increment Bd
X	r	-6	00 r 0110	RAM(B) ↔ A Br ⊕ r → Br	None	Exchange RAM with A, Exclusive-OR Br with r
XAD	r,d	23 --	0010 0011 10 r d	RAM(r,d) ↔ A	None	Exchange A with RAM pointed to directly by r,d
XDS	r	-7	00 r 0111	RAM(B) ↔ A Bd - 1 → Bd Br ⊕ r → Br	Bd decrements past 0	Exchange RAM with A and Decrement Bd, Exclusive-OR Br with r
XIS	r	-4	00 r 0100	RAM(B) ↔ A Bd + 1 → Bd Br ⊕ r → Br	Bd increments past 15	Exchange RAM with A and Increment Bd, Exclusive-OR Br with r
<b>REGISTER REFERENCE INSTRUCTIONS</b>						
CAB		50	0101 0000	A → Bd	None	Copy A to Bd
CBA		4E	0100 1110	Bd → A	None	Copy Bd to A
LBI	r,d	--	00 r (d-1) (d=0, 9:15) or 0011 0011 10 r d (any d)	r,d → B	Skip until not a LBI	Load B Immediate with r,d (Note 6)
LEI	y	33 6-	0011 0011 0110 y	y → EN	None	Load EN Immediate (Note 7)
XABR		12	0001 0010	A ↔ Br (0,0 → A <sub>3</sub> ,A <sub>2</sub> )	None	Exchange A with Br
<b>TEST INSTRUCTIONS</b>						
SKC		20	0010 0000		C = "1"	Skip if C is True
SKE		21	0010 0001		A = RAM(B)	Skip if A Equals RAM
SKGZ		33 21	0011 0011 0010 0001		G <sub>3:0</sub> = 0	Skip if G is Zero (all 4 bits)
SKGBZ		33	0011 0011	1st byte		Skip if G Bit is Zero
	0	01	0000 0001	} 2nd byte	G <sub>0</sub> = 0	
	1	11	0001 0001		G <sub>1</sub> = 0	
	2	03	0000 0011		G <sub>2</sub> = 0	
	3	13	0001 0011		G <sub>3</sub> = 0	
SKMBZ		0 1 2 3	0000 0001 0001 0001 0000 0011 0001 0011		RAM(B) <sub>0</sub> = 0 RAM(B) <sub>1</sub> = 0 RAM(B) <sub>2</sub> = 0 RAM(B) <sub>3</sub> = 0	Skip if RAM Bit is Zero
SKT		41	0100 0001		A time-base counter carry has occurred since last test	Skip on Timer (Note 3)

Table 2. COP402/COP402M Instruction Set Table (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
INPUT/OUTPUT INSTRUCTIONS						
ING		33	0011 0011	G → A	None	Input G Ports to A
		2A	0010 1010			
ININ		33	0011 0011	IN → A	None	Input IN Inputs to A (Notes 2 and 8)
		28	0010 1000			
INIL		33	0011 0011	IL <sub>3</sub> , "0", IL <sub>0</sub> → A	None	Input IL Latches to A (Note 3)
		29	0010 1001			
INL		33	0011 0011	L <sub>7,4</sub> → RAM(B) L <sub>3,0</sub> → A	None	Input L Ports to RAM,A
		2E	0010 1110			
OBD		33	0011 0011	Bd → D	None	Output Bd to D Outputs
		3E	0011 1110			
OGI	y	33	0011 0011	y → G	None	Output to G Ports Immediate
		5-	0101  y			
OMG		33	0011 0011	RAM(B) → G	None	Output RAM to G Ports
		3A	0011 1010			
XAS		4F	0100 1111	A ↔ SIO, C → SKL	None	Exchange A with SIO (Note 3)

**Note 1:** All subscripts for alphabetical symbols indicate bit numbers unless explicitly defined (e.g., Br and Bd are explicitly defined). Bits are numbered 0 to N where 0 signifies the least significant bit (low-order, right-most bit). For example, A<sub>3</sub> indicates the most significant (left-most) bit of the 4-bit A register.

**Note 2:** The ININ instruction is not available on the 24-pin COP421 since this device does not contain the IN inputs.

**Note 3:** For additional information on the operation of the XAS, JID, LQID, INIL, and SKT instructions, see below.

**Note 4:** The JP instruction allows a jump, while in subroutine pages 2 or 3, to any ROM location within the two-page boundary of pages 2 or 3. The JP instruction, otherwise, permits a jump to a ROM location within the current 64-word page. JP may not jump to the last word of a page.

**Note 5:** A JSRP transfers program control to subroutine page 2 (0010 is loaded into the upper 4 bits of P). A JSRP may not be used when in pages 2 or 3. JSRP may not jump to the last word in page 2.

**Note 6:** LBI is a single-byte instruction if d = 0, 9, 10, 11, 12, 13, 14, or 15. The machine code for the lower 4 bits equals the binary value of the "d" data *minus 1*, e.g., to load the lower four bits of B (Bd) with the value 9 (1001<sub>2</sub>), the lower 4 bits of the LBI instruction equal 8 (1000<sub>2</sub>). To load 0, the lower 4 bits of the LBI instruction should equal 15 (1111<sub>2</sub>).

**Note 7:** Machine code for operand field y for LEI instruction should equal the binary value to be latched into EN, where a "1" or "0" in each bit of EN corresponds with the selection or deselection of a particular function associated with each bit. (See Functional Description, EN Register.)

**Note 8:** The COP402M will always read a "1" into A1 with the ININ instruction.

## INIL Instruction

INIL (Input IL Latches to A) inputs 2 latches,  $IL_3$  and  $IL_0$  (see Figure 12) and CKO into A. The  $IL_3$  and  $IL_0$  latches are set if a low-going pulse ("1" to "0") has occurred on the  $IN_3$  and  $IN_0$  inputs since the last INIL instruction, provided the input pulse stays low for at least two instruction times. Execution of an INIL inputs  $IL_3$  and  $IL_0$  into A3 and A0 respectively, and resets these latches to allow them to respond to subsequent low-going pulses on the  $IN_3$  and  $IN_0$  lines. If CKO is mask programmed as a general purpose input, an INIL will input the state of CKO into A2. If CKO has not been so programmed, a "1" will be placed in A2. A "0" is always placed in A1 upon the execution of an INIL. The general purpose inputs  $IN_3$ – $IN_0$  are input to A upon the execution of an ININ instruction. (See Table 2, ININ Instruction.) INIL is useful in recognizing pulses of short duration or pulses which occur too often to be read conveniently by an ININ instruction.

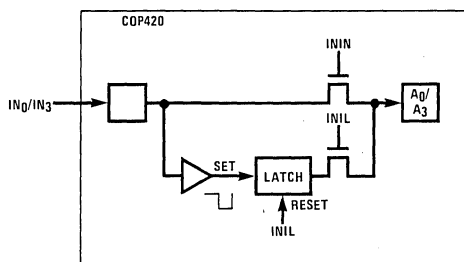


Figure 12.  $IN_0/IN_3$  Latches

## LQID Instruction

LQID (Load Q Indirect) loads the 8-bit Q register with the contents of ROM pointed to by the 10-bit word  $PC_9$ ,  $PC_8$ , A, M. LQID can be used for table lookup or code conversion such as BCD to seven-segment. The LQID instruction "pushes" the stack ( $PC + 1 \rightarrow SA \rightarrow SB \rightarrow SC$ ) and replaces the least significant 8 bits of PC as follows:  $A \rightarrow PC_{7:4}$ ,  $RAM(B) \rightarrow PC_{3:0}$ , leaving  $PC_9$  and  $PC_8$  unchanged. The ROM data pointed to by the new address is fetched and loaded into the Q latches. Next, the stack is "popped" ( $SC \rightarrow SB \rightarrow SA \rightarrow PC$ ), restoring the saved value of PC to continue sequential program execution. Since LQID pushes  $SB \rightarrow SC$ , the previous contents of SC are lost. Also, when LQID pops the stack, the previously pushed contents of SB are left in SC. The net result is that the contents of SB are placed in SC ( $SB \rightarrow SC$ ). Note that LQID takes two instruction cycle times to execute.

## SKT Instruction

The SKT (Skip on Timer) instruction tests the state of an internal 10-bit time-base counter. This counter divides the instruction cycle clock frequency by 1024 and provides a latched indication of counter overflow. The SKT instruction tests this latch, executing the next program instruction if the latch is

not set. If the latch has been set since the previous test, the next program instruction is skipped and the latch is reset. The features associated with this instruction, therefore, allow the controller to generate its own time-base for real-time processing rather than relying on an external input signal.

For example, using a 2.097MHz crystal as the time-base to the clock generator, the instruction cycle clock frequency will be 131kHz (crystal frequency  $\div$  16) and the binary counter output pulse frequency will be 128Hz. For time-of-day or similar real-time processing, the SKT instruction can call a routine which increments a "seconds" counter every 128 ticks.

## Instruction Set Notes

- The first word of a program (ROM address 0) must be a CLRA (Clear A) instruction.
- Although skipped instructions are not executed, one instruction cycle time is devoted to skipping each byte of the skipped instruction. Thus all program paths take the same number of cycle times whether instructions are skipped or executed, except JID and LQID. LQID and JID take two cycle times if executed and one if skipped.
- The ROM is organized into 16 pages of 64 words each. The Program Counter is a 10-bit binary counter, and will count through page boundaries. If a JP, JSRP, JID or LQID instruction is located in the last word of a page, the instruction operates as if it were in the next page. For example: a JP located in the last word of a page will jump to a location in the next page. Also, a LQID or JID located in the last word of page 3, 7, 11, or 15 will access data in the next group of 4 pages.

## Typical Application: PROM-Based System

The COP402 may be used to exactly emulate the COP420. Figure 12 shows the interconnect to implement a COP420 hardware emulation. This connection uses two MM5204 EPROMs as external memory. Other memory can be used such as bipolar PROM or RAM.

Pins IP7–IP0 are bidirectional inputs and outputs. When the  $AD/\overline{DATA}$  clocking output turns on, the EPROM drivers are disabled and IP7–IP0 output addresses. The 8-bit latch (MM74C373) latches the addresses to drive the memory.

When  $AD/\overline{DATA}$  turns off, the EPROMs are enabled and the IP7–IP0 pins will input the memory data. P8 and P9 output the most significant address bits to the memory. (SKIP output may be used for program debug if needed.)

The other 28 pins of the COP402 may be configured exactly the same as a COP420. The COP402M chip can be used if the MICROBUS™ feature of the COP420 is needed.

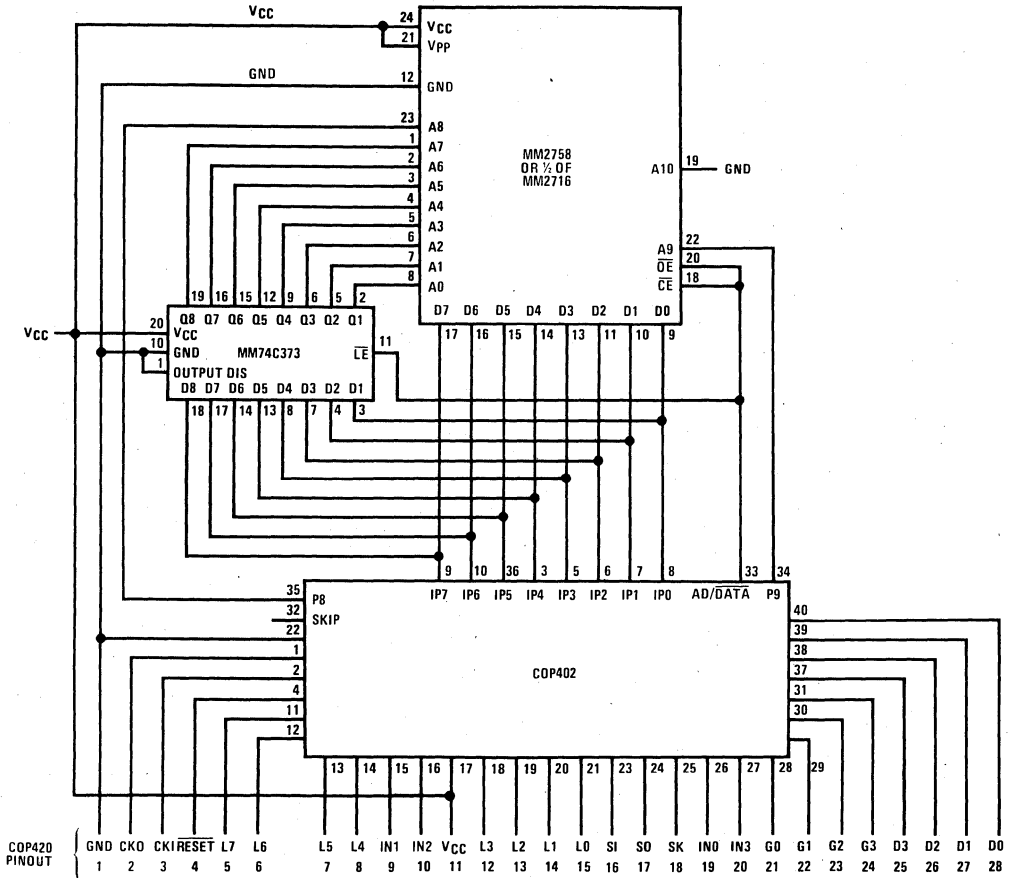


Figure 13. COP402 Used to Emulate a COP420

## COP402 Mask Options

The following COP402 options have been implemented in this basic version of the COP402. Subsequent versions of the COP402 will implement different combinations of available options; such versions will be identified as COP402-A, COP402-B, etc.

Option Value	Comment
Option 1 = 0	Ground Pin — no option available
Option 2 = 0	CKO is clock generator output to crystal
Option 3 = 0	CKI is crystal input $\pm 16$ (may be overridden externally)
Option 4 = 0	RESET pin has load device to $V_{CC}$
Option 5 = 2 (402)	L7 has LED direct-drive output
= 3 (402M)	L7 has TRI-STATE® push-pull output
Option 6 = 2,3	L6 same as L7
Option 7 = 2,3	L5 same as L7
Option 8 = 2,3	L4 same as L7
Option 9 = 0 (402)	IN1 has load device to $V_{CC}$
= 1 (402M)	Hi Z
Option 10 = 0 (402)	IN2 has load device to $V_{CC}$
= 1 (402M)	Hi Z
Option 11 = 0	$V_{CC}$ pin — no option available
Option 12 = 2,3	L3 same as L7
Option 13 = 2,3	L2 same as L7
Option 14 = 2,3	L1 same as L7
Option 15 = 2,3	L0 same as L7
Option 16 = 0	SI has load device to $V_{CC}$
Option 17 = 2	SO has push-pull output
Option 18 = 2	SK has push-pull output
Option 19 = 0	IN0 has load device to $V_{CC}$
Option 20 = 0 (402)	IN3 has load device to $V_{CC}$
= 1 (402M)	Hi Z
Option 21 = 0	G0 has standard output
Option 22 = 0	G1 same as G0
Option 23 = 0	G2 same as G0
Option 24 = 0	G3 same as G0
Option 25 = 0	D3 has standard output
Option 26 = 0	D2 same as D3
Option 27 = 0	D1 same as D3
Option 28 = 0	D0 same as D3
Option 29 = 0 (402)	normal operation
= 1 (402M)	MICROBUS™ operation
Option 30 = N/A	40-pin package



## COP404/COP304 ROMless N-Channel Microcontrollers

### General Description

The COP404/COP304 ROMless N-Channel Microcontrollers are members of the COPSTM family, fabricated using N-channel, silicon gate MOS technology. Each microcontroller contains all system timing, internal logic, RAM and I/O necessary to implement dedicated control functions in a variety of applications, and is identical to the COP440/COP340 devices, except that the ROM has been removed; pins have been added to output the ROM address and to input ROM data. In a system, the COP404 will perform exactly as the COP440; this important benefit facilitates development and debug of a COP440 program prior to masking the final part. Features include single supply operation, various output configurations, and an instruction set, internal architecture, and I/O scheme designed to facilitate keyboard input, display output and data manipulation. Standard test procedures and reliable high-density fabrication techniques provide the medium to large volume customers with a controller-oriented processor at a low end-product cost. COP304 is an exact functional equivalent version of COP404, but with an extended temperature range (-40°C to +85°C).

### Features

- Exact circuit equivalent of COP440
- Standard 48-pin dual-in-line package
- Interfaces with standard PROM or ROM
- Enhanced, more powerful instruction set
- 160 × 4 RAM, addresses up to 2k × 8 ROM
- MICROBUS™ compatible
- Zero-crossing detect circuitry with hysteresis
- True multi-vectored interrupt from four selectable sources (plus restart)
- Four-level subroutine stack (in RAM)
- 4 μs cycle time
- Single supply operation (4.5V–6.3V)
- Programmable time-base counter for real-time processing
- Internal binary counter/register with MICROWIRE™ compatible serial I/O
- General purpose and TRI-STATE® outputs
- TTL/CMOS compatible in and out
- Software/hardware compatible with other members of COP400 family
- Extended temperature range device COP304 (-40°C to +85°C)
- Compatible dual CPU device available

TRI-STATE is a registered trademark of National Semiconductor Corp.  
COPS, MICROBUS, and MICROWIRE, are trademarks of National Semiconductor Corp.

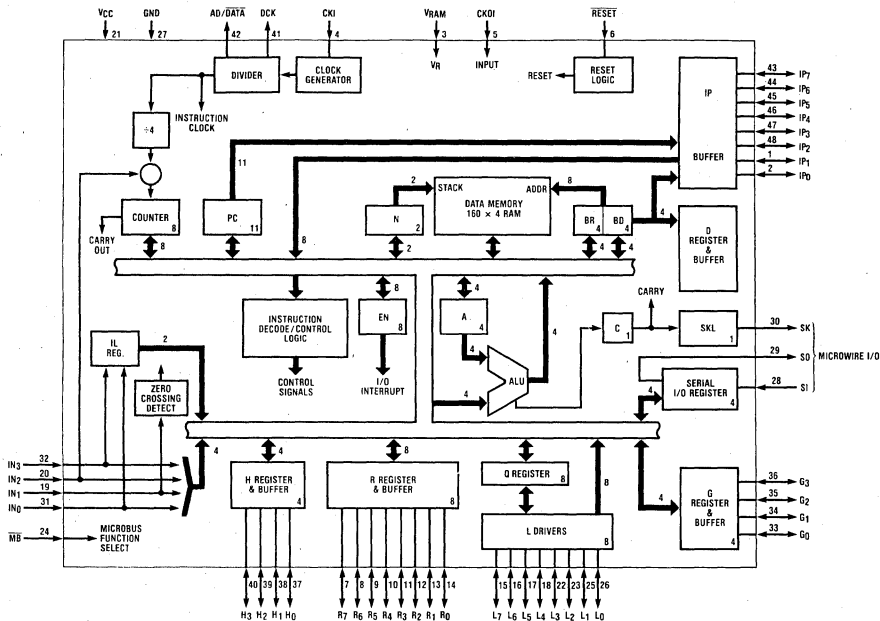


Figure 1. COP404 Block Diagram

**COP404****Absolute Maximum Ratings**

Voltage at Zero-Crossing Detect Pin Relative to GND	-1.2V to +15V
Voltage at Any Other Pin Relative to GND	-0.5V to +7V
Ambient Operating Temperature	0°C to +70°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C
Power Dissipation	0.75 Watt at 25°C 0.4 Watt at 70°C
Total Source Current	150 mA
Total Sink Current	90 mA

*Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

**DC Electrical Characteristics** 0°C ≤ T<sub>A</sub> ≤ +70°C, 4.5V ≤ V<sub>CC</sub> ≤ 6.3V unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Operating Voltage (V <sub>CC</sub> )	Note 3	4.5	6.3	V
Power Supply Ripple	(peak to peak)		0.4	V
Operating Supply Current	(All inputs and outputs open) T <sub>A</sub> = 0°C T <sub>A</sub> = 25°C T <sub>A</sub> = 70°C		44 37 30	mA mA mA
V <sub>R</sub> RAM Power Supply Current	V <sub>R</sub> = 3.3V		3	mA
Input Voltage Levels				
CKI Input Levels (≠16)				
Logic High (V <sub>IH</sub> )	V <sub>CC</sub> = Max.,	2.5		V
Logic High (V <sub>IH</sub> )	V <sub>CC</sub> = 5V ± 5%	2.0		V
Logic Low (V <sub>IL</sub> )		-0.3	0.4	V
RESET Input Levels	(Schmitt Trigger Input)			
Logic High		0.7V <sub>CC</sub>		V
Logic Low		-0.3	0.6	V
Zero-Crossing Detect Input (IN <sub>1</sub> )	Zero-Crossing Interrupt Input; INIL Instruction			
Trip Point		-0.15	0.15	V
Logic High (V <sub>IH</sub> ) Limit			12	V
Logic Low (V <sub>IL</sub> ) Limit		-0.8		V
IN <sub>1</sub>				
Logic High	Interrupt Input; ININ Instruction;	3.0		V
Logic Low	MICROBUS™ Input	-0.3	0.8	V
All Other Inputs				
Logic High	V <sub>CC</sub> = Max.	2.5		V
Logic High	V <sub>CC</sub> = 5V ± 5%	2.0		V
Logic Low		-0.3	0.8	V
IN <sub>1</sub> Input Resistance to Ground	V <sub>IH</sub> = 1.0V	1.5	4.6	kΩ
Input Load Source Current	V <sub>IH</sub> = 2.0V, V <sub>CC</sub> = 4.5V	14	230	μA
Input Capacitance			7.0	pF
Hi-Z Input Leakage		-1.0	+1.0	μA



# COP404

## DC Electrical Characteristics (Cont'd)

Parameter	Conditions	Min.	Max.	Units
Output Voltage Levels				
Standard Output				
TTL Operation				
Logic High ( $V_{OH}$ )	$I_{OH} = -100\mu A$	2.4		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 1.6\text{mA}$		0.4	V
CMOS Operation				
Logic High ( $V_{OH}$ )	$I_{OH} = -10\mu A$	$V_{CC} - 0.4$		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 10\mu A$		0.2	V
TRI-STATE® Output				
TTL Operation				
Logic High ( $V_{OH}$ )	$I_{OH} = -100\mu A$	2.4		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 1.6\text{mA}$		0.4	V
CMOS Operation	$33\text{k}\Omega \geq R_L \geq 4.7\text{k}\Omega$			
Logic High ( $V_{OH}$ )	$I_{OH} = -10\mu A$	$V_{CC} - 0.5$		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 1.6\text{mA}$		0.4	V
Output Current Levels				
Standard Output Source Current	$V_{CC} = 4.5\text{V}, V_{OH} = 2.4\text{V}$	-100	-650	$\mu A$
TRI-STATE Output Leakage Current		-2.5	+2.5	$\mu A$
Total Sink Current Allowed				
All I/O Combined			90	mA
Each L, R Port			20	mA
Each D, G, H Port			10	mA
SO, SK			2.5	mA
IP			1.8	mA
Total Source Current Allowed	Note 4			
All I/O Combined			150	mA
L Port			120	mA
L <sub>7</sub> -L <sub>4</sub>			70	mA
L <sub>3</sub> -L <sub>0</sub>			70	mA
Each L Pin			23	mA
All Other Output Pins			1.6	mA

**COP304****Absolute Maximum Ratings**

Voltage at Zero-Crossing Detect Pin Relative to GND	-1.2V to +15V
Voltage at Any Other Pin Relative to GND	-0.5V to +7V
Ambient Operating Temperature	-40°C to +85°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C
Power Dissipation	0.75 Watt at 25°C 0.25 Watt at 85°C
Total Source Current	150 mA
Total Sink Current	90 mA

*Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

**DC Electrical Characteristics**  $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{\text{CC}} \leq 5.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Operating Voltage ( $V_{\text{CC}}$ )	Note 3	4.5	5.5	V
Power Supply Ripple	(peak to peak)		0.4	V
Operating Supply Current	(All inputs and outputs open) $T_A = -40^{\circ}\text{C}$ $T_A = 25^{\circ}\text{C}$ $T_A = 85^{\circ}\text{C}$		57 37 29	mA mA mA
$V_{\text{R}}$ RAM Power Supply Current	$V_{\text{R}} = 3.3\text{V}$		4	mA
Input Voltage Levels				
CKI Input Levels ( $\pm 16$ )				
Logic High ( $V_{\text{IH}}$ )		2.2		V
Logic Low ( $V_{\text{IL}}$ )		-0.3	0.3	V
RESET Input Levels	(Schmitt Trigger Input)			
Logic High		$0.7V_{\text{CC}}$		V
Logic Low		-0.3	0.4	V
Zero Crossing Detect Input ( $\text{IN}_1$ )	Zero Crossing Interrupt Input; INIL Instruction			
Trip Point		-0.15	0.15	V
Logic High ( $V_{\text{IH}}$ ) Limit			12	V
Logic Low ( $V_{\text{IL}}$ ) Limit		-0.8		V
$\text{IN}_1$				
Logic High	Interrupt Input; ININ Instruction;	3.3		V
Logic Low	MICROBUS™ Input	-0.3	0.6	V
All Other Inputs				
Logic High		2.2		V
Logic Low		-0.3	0.6	V
$\text{IN}_1$ Input Resistance to Ground	$V_{\text{IH}} = 1.0\text{V}$	1.4	4.6	k $\Omega$
Input Load Source Current	$V_{\text{IH}} = 2.0\text{V}$ , $V_{\text{CC}} = 4.5\text{V}$	14	230	$\mu\text{A}$
Input Capacitance			7.0	pF
Hi-Z Input Leakage		-2.0	+2.0	$\mu\text{A}$



# COP304

## DC Electrical Characteristics (Cont'd)

Parameter	Conditions	Min.	Max.	Units
Output Voltage Levels				
Standard Output				
TTL Operation				
Logic High ( $V_{OH}$ )	$I_{OH} = -100\mu A$	2.4		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 1.6mA$		0.4	V
CMOS Operation				
Logic High ( $V_{OH}$ )	$I_{OH} = -10\mu A$	$V_{CC} - 0.5$		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 10\mu A$		0.2	V
TRI-STATE® Output				
TTL Operation				
Logic High ( $V_{OH}$ )	$I_{OH} = -100\mu A$	2.2		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 1.6mA$		0.4	V
CMOS Operation	$33k\Omega \geq R_L \geq 4.7k\Omega$			
Logic High ( $V_{OH}$ )	$I_{OH} = -10\mu A$	$V_{CC} - 0.7$		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 1.6mA$		0.4	V
Output Current Levels				
Standard Output Source Current	$V_{CC} = 4.5V, V_{OH} = 2.4V$	-100	-800	$\mu A$
TRI-STATE Output Leakage Current		-5.0	+5.0	$\mu A$
Total Sink Current Allowed				
All I/O Combined			75	mA
Each L, R Port			20	mA
Each D, G, H Port			10	mA
SO, SK			2.5	mA
IP			1.8	mA
Total Source Current Allowed	Note 4			
All I/O Combined			150	mA
L Port			120	mA
L <sub>7</sub> -L <sub>4</sub>			70	mA
L <sub>3</sub> -L <sub>0</sub>			70	mA
Each L Pin			23	mA
All Other Output Pins			1.6	mA

## AC Electrical Characteristics

**COP404:**  $0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 6.3\text{V}$  unless otherwise noted.

**COP304:**  $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 5.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Instruction Cycle Time — $t_E$		4.0	10	$\mu\text{s}$
CKI Frequency	+16 mode	1.6	4.0	MHz
Duty Cycle (Note 1)	$f_i = 4\text{ MHz}$	30	60	%
Rise Time	$f_i = 4\text{ MHz}$		60	ns
Fall Time	$f_i = 4\text{ MHz}$		40	ns
INPUTS: (Figure 3)				
SI				
$t_{\text{SETUP}}$		0.3		$\mu\text{s}$
$t_{\text{HOLD}}$		300		ns
IP				
$t_{\text{SETUP}}$		0.25		$\mu\text{s}$
$t_{\text{HOLD}}$		250		ns
$t_{\text{HOLD}}$	From $\overline{\text{AD}}/\overline{\text{DATA}}$ rising edge	0		ns
All Other Inputs				
$t_{\text{SETUP}}$		1.7		$\mu\text{s}$
$t_{\text{HOLD}}$		300		ns
OUTPUT PROPAGATION DELAY	Test Condition: $C_L = 50\text{ pF}$ , $V_{\text{OUT}} = 1.5\text{V}$			
IP			1.94	$\mu\text{s}$
$t_{\text{pd1A}}$ , $t_{\text{pd0A}}$			0.94	$\mu\text{s}$
$t_{\text{pd1B}}$ , $t_{\text{pd0B}}$				
DCK			375	ns
$t_{\text{pd1}}$ , $t_{\text{pd0}}$				
AD/DATA			300	ns
$t_{\text{pd1}}$ , $t_{\text{pd0}}$				
SO, SK				
$t_{\text{pd1}}$ , $t_{\text{pd0}}$	$R_L = 2.4\text{ k}\Omega$		1.0	$\mu\text{s}$
All Other Outputs	$R_L = 5.0\text{ k}\Omega$		1.4	$\mu\text{s}$
MICROBUS™ TIMING	$C_L = 100\text{ pF}$ , $V_{CC} = 5\text{V} \pm 5\%$ TRI-STATE® outputs			
Read Operation				
Chip Select Stable Before $\overline{\text{RD}}$ — $t_{\text{CSR}}$		65		ns
Chip Select Hold Time for $\overline{\text{RD}}$ — $t_{\text{RCS}}$		20		ns
$\overline{\text{RD}}$ Pulse Width— $t_{\text{RR}}$		400		ns
Data Delay from $\overline{\text{RD}}$ — $t_{\text{RD}}$			375	ns
$\overline{\text{RD}}$ to Data Floating— $t_{\text{DF}}$			250	ns
Write Operation				
Chip Select Stable Before $\overline{\text{WR}}$ — $t_{\text{CSW}}$		65		ns
Chip Select Hold Time for $\overline{\text{WR}}$ — $t_{\text{WCS}}$		20		ns
$\overline{\text{WR}}$ Pulse Width— $t_{\text{WW}}$		400		ns
Data Set-Up Time for $\overline{\text{WR}}$ — $t_{\text{DW}}$		320		ns
Data Hold Time for $\overline{\text{WR}}$ — $t_{\text{WD}}$		100		ns
INTR Transition Time from $\overline{\text{WR}}$ — $t_{\text{WI}}$			700	ns

**Note 1:** Duty Cycle =  $t_{\text{WI}} / (t_{\text{WI}} + t_{\text{WO}})$ .

**Note 2:** See Figure for additional I/O Characteristics.

**Note 3:**  $V_{CC}$  voltage change must be less than 0.5V in a 1ms period to maintain proper operation.

**Note 4:** Exercise great care not to exceed maximum device power dissipation limits when direct-driving LEDs (or sourcing similar loads) at high temperature.

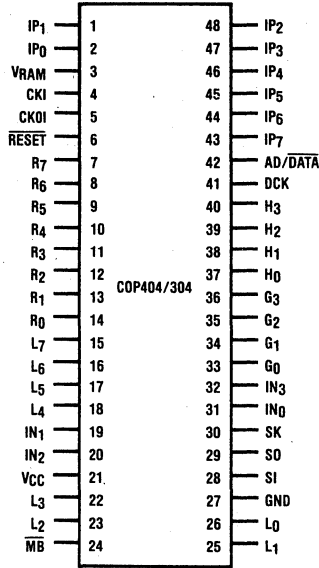


Figure 2. Connection Diagram

Order Number COP404N, COP304N  
NS Package N48A

Pin	Description
L7-L0	8-bit bidirectional TRI-STATE® I/O port
G3-G0	4-bit bidirectional I/O port
IN3-IN0	4-bit general purpose input port
H3-H0	4-bit bidirectional I/O port
R7-R0	8-bit bidirectional TRI-STATE I/O port
SI	Serial input
SO	Serial output (or general purpose output)
SK	Logic-controlled clock (or general purpose output)
CKI	System oscillator input
CKOI	General purpose input
VRAM	Power supply to first 4 registers of RAM
MB	MICROBUS™ function select
DCK	Clock output to latch D outputs and high order address bits
AD/DATA	Address out/data in flag
IP1-IP0	8-bit bidirectional port for ROM address, ROM data and D outputs
RESET	System reset input
VCC	Power Supply
GND	Ground

### Timing Diagram

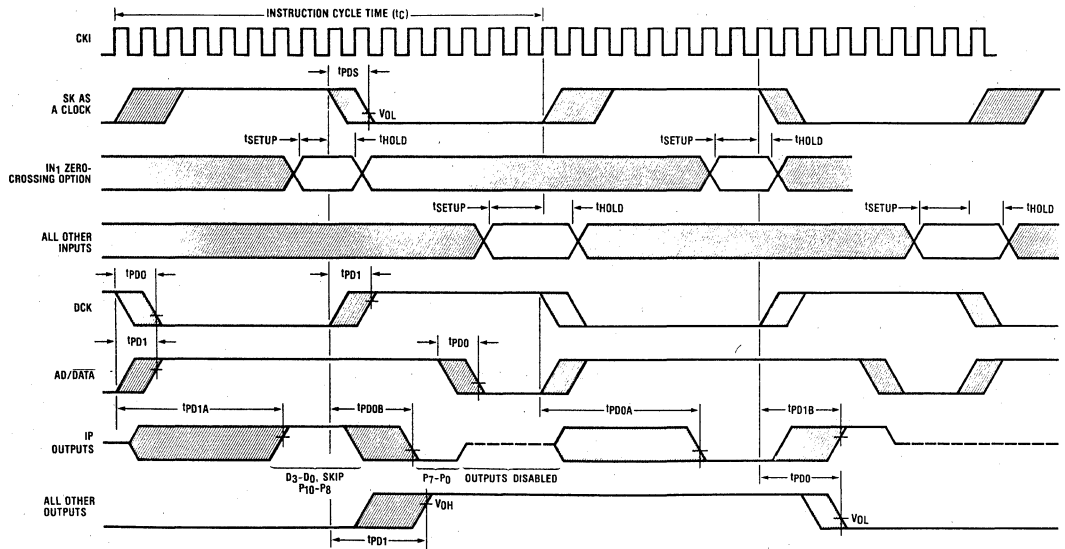


Figure 3. Input/Output Timing Diagrams (+16 Mode)

## Functional Description

The COP404 is a ROMless microcontroller for emulating the COP440 or for stand-alone applications. Please refer to the COP440 description for detail functional description. The following describes functions that are unique to the COP404 or are different from those in COP440. All references to COP404 also apply to COP304. *Figures 1 and 2* show the COP404 block diagram and pin-out.

### Program Memory

Program memory consists of 2048 bytes of external memory (on-chip in the COP440) that can be accessed through the IP port. See External Memory Interface below.

### D Port

The D3-D0 outputs are missing from this 48-pin package, but may be recovered through the IP port (see External Memory Interface below). Note that the recovered signals have the same timing but different output drive capability as those from the COP440 (see D Port Characteristics below).

### MICROBUS™ and Zero-Crossing Detect Input Option

The MICROBUS compatible I/O, selected by a mask option on the COP440, is selected by tying the  $\overline{MB}$  pin directly to ground. When the MICROBUS compatible I/O is not desired, the  $\overline{MB}$  pin should be tied to  $V_{CC}$ . Note that none of the IN inputs are Hi-Z. Since zero-crossing detect input (used by INIL instruction and zero-crossing interrupt feature) is chosen for IN1, the IN1 input "1" level for ININ instruction, IN1 interrupt, and MICROBUS input is 3V. Even though the MICROBUS option and zero-crossing detector option appear on the COP404, they are mutually exclusive on the COP440.

### Oscillator

CKI is an external clock input signal. The clock frequency is divided by 16 to give the execution frequency.

### CKO Pin Options

Two different CKO functions of the COP440 are available on the COP404.  $V_{RAM}$  supplies power to the lower four registers of RAM, and CKO1 is an interrupt input or a general purpose input, reading into bit 2 of A (accumulator) through the INIL instruction.

### External Memory Interface

The COP404 is designed for use with an external program memory. This memory may be implemented using any devices having the following characteristics:

1. Random addressing
2. TTL-compatible TRI-STATE® outputs
3. TTL-compatible inputs
4. Access time = 450ns maximum

Typically these requirements are met using bipolar or MOS PROMs.

*Figure 3* shows the timings for IP port and the external memory interface clocks—DCK and AD/DATA. While DCK is low, the upper three address bits, P10-P8, of the next instruction to be executed appear at IP2-IP0 respectively; D3-D0 appear at IP7-IP4 and IP3 contains the SKIP output used by the COPST™ Program Development System (PDS). The rising edge of DCK clocks these data into D flip-flops, e.g., 74LS374. The timing of D port data is then the same for COP404 and COP440. After DCK has risen to a "1" level, the remaining address bits (P7-P0) appear at IP7-IP0. The falling edge of AD/DATA latches these data into flow-through latches, e.g., 74LS373. The latched addresses provide the inputs to the external memory. When AD/DATA goes low, the IP outputs are disabled and the IP lines become program memory inputs from the external memory. Note that DCK has a duty cycle of about 50% and AD/DATA has a duty cycle of about 75%. *Figure 4* shows how to emulate the COP440 using a COP404 and an EPROM as the external memory.

### I/O Options

All inputs except IN1 and CKI have on-chip depletion load devices to  $V_{CC}$ . IN1 has a resistive load to GND due to the zero-crossing input. CKI is a Hi-Z input.

G and H ports have standard outputs. L and R ports have TRI-STATE outputs. IP port, DCK, AD/DATA, SO and SK have push-pull outputs.

### LED Drive

The TRI-STATE outputs of L port may be used to drive the segments of an LED display. External current limiting resistors of 100 ohms must be connected between the L outputs and the LED segments.

### D Port Characteristics

Since the D port is recovered through an external latch, the output drive is that of the latch and not that of COP440. Using the set-up as shown in *Figure 4*, at an output "0" level of 0.4V, the 74LS374 may sink 10 times as much current as the COP440. At an output "1" level of 2.4V, the 74LS374 may source 10 times as much current as the COP440. On the other hand, the output "1" level of 74LS374 latch does not go to  $V_{CC}$  without an external pull-up resistor. In order to better approximate the COP440 output characteristics, add a 74C906 buffer to the output of the 74LS374, thus emulating an open drain D output. A pull-up resistor of 10k should be added to the input of the buffer. To emulate the standard output, add a pull-up resistor between 2.7k and 15k to the output of the 74C906.

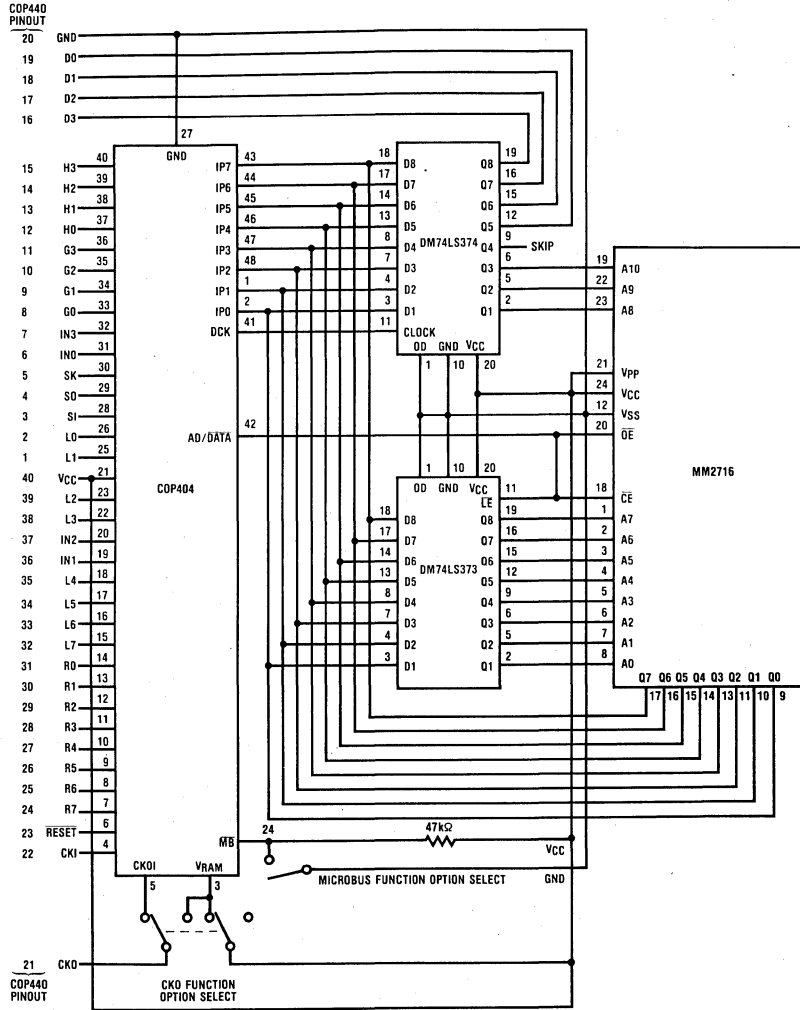


Figure 4. COP204 Used to Emulate a COP440

**COP404 Mask Options**

The following COP440 options have been implemented in the COP404.

Option Value	Comment
Option 1- 2=3	L outputs are TRI-STATE®
Option 3 =0	SI has load to $V_{CC}$
Option 4 =2	SO is push-pull output
Option 5 =2	SK is push-pull output
Option 6 =0	IN0 has load to $V_{CC}$
Option 7 =0	IN3 has load to $V_{CC}$
Option 8-11=0	G outputs are standard
Option 12-15=0	H outputs are standard
Option 16-19=N/A	D outputs are derived from external latch, see <i>Figure 4</i>
Option 20 = N/A	GND—No option
Option 21 = 1,2	CKO is replaced by $V_{RAM}$ and CKOI
Option 22 =0	CKI is input clock divided by 16
Option 23 =0	$\overline{RESET}$ has load to $V_{CC}$
Option 24-31=3	R outputs are TRI-STATE
Option 32-35=3	L outputs are TRI-STATE
Option 36 =2	IN1 is zero-crossing detect input
Option 37 =0	IN2 has load to $V_{CC}$
Option 38-39=3	L outputs are TRI-STATE
Option 40 = N/A	$V_{CC}$ —No option available
Option 41 =0,1	MICROBUS™ option is pin selectable
Option 42-48=0	Inputs have standard TTL levels
Option 49 = N/A	No option available
Option 50 = N/A	48-pin package





# COP404L/COP304L ROMless N-Channel Microcontrollers

## General Description

The COP404L ROMless Microcontroller is a member of the COPS™ family, fabricated using N-channel, silicon gate MOS technology. The COP404L contains CPU, RAM, I/O and is identical to a COP444L device except the ROM has been removed and pins have been added to output the ROM address and to input the ROM data. In a system the COP404L will perform exactly as the COP444L. This important benefit facilitates development and debug of a COP program prior to masking the final part. The COP404L is also appropriate in low volume applications, or when the program might be changing. The COP404L may be used to emulate the COP444L, COP445L, COP420L, and the COP421L.

The COP304L is an exact functional equivalent of the COP404L, but with extended temperature range.

## Features

- Exact circuit equivalent of COP444L
- Low cost
- Powerful instruction set
- 128 x 4 RAM, addresses 2048 x 8 ROM
- True vectored interrupt, plus restart
- Three-level subroutine stack
- 15µs instruction time
- Single supply operation (4.5-9.5V)
- Low current drain (16mA max.)
- Internal time-base counter for real-time processing
- Internal binary counter register with MICROWIRE™ compatible serial I/O
- General purpose outputs
- LSTTL/CMOS compatible in and out
- Direct drive of LED digit and segment lines
- Software/hardware compatible with other members of COP400 family
- Extended temperature range (-40°C to +85°C) device COP304L

COPS and MICROWIRE are trademarks of National Semiconductor Corp.

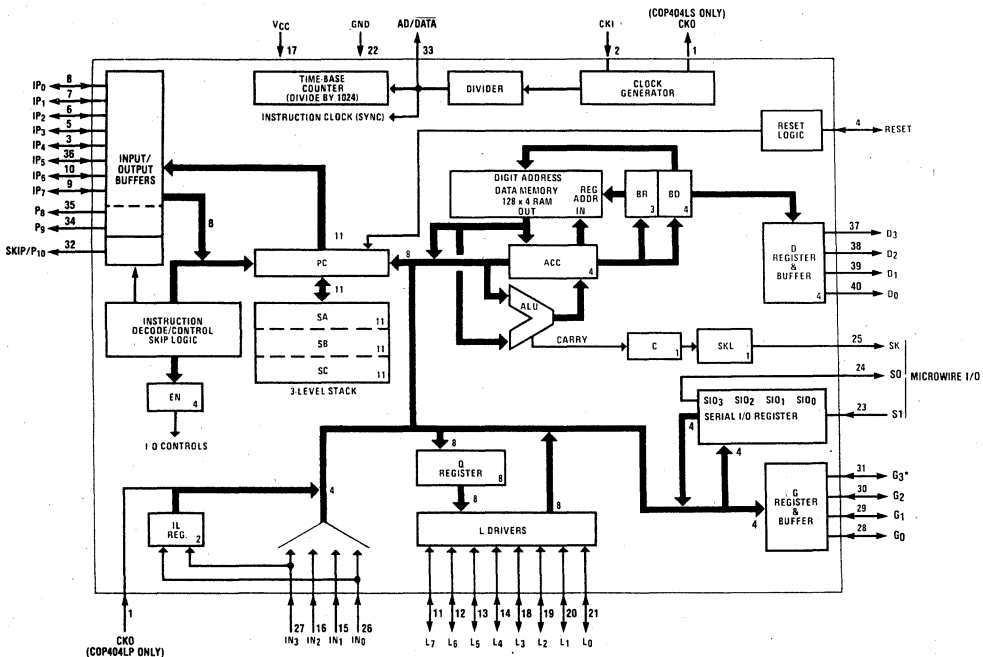


Figure 1. COP404L Block Diagram

**COP404L****Absolute Maximum Ratings**

Voltage at Any Pin Relative to GND	-0.5V to +10V
Ambient Operating Temperature	0°C to +70°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C
Power Dissipation	0.75 Watt at 25°C 0.4 Watt at 70°C
Total Source Current	120 mA
Total Sink Current	140 mA

*Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

**DC Electrical Characteristics**  $0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 9.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Operating Voltage ( $V_{CC}$ )	(Note 2)	4.5	9.5	V
Power Supply Ripple	peak to peak		0.5	V
Operating Supply Current	all inputs and outputs open		16	mA
Input Voltage Levels				
CKI Input Levels				
Crystal Input				
Logic High ( $V_{IH}$ )		2.0		V
Logic Low ( $V_{IL}$ )		-0.3	0.4	V
$\overline{\text{RESET}}$ Input Levels	Schmitt Trigger Input			
Logic High		$0.7 V_{CC}$		V
Logic Low		-0.3	0.6	V
IP0-IP7, SI Input Levels				
Logic High	$V_{CC} = 9.5\text{V}$	2.4		V
Logic High	$V_{CC} = 5\text{V} \pm 5\%$	2.0		V
Logic Low		-0.3	0.8	V
All Other Inputs				
Logic High	high trip level options selected	3.6		V
Logic Low		-0.3	1.2	V
Input Capacitance			7	pF
Output Voltage Levels				
LSTTL Operation				
Logic High ( $V_{OH}$ )	$V_{CC} = 5\text{V} \pm 5\%$ $I_{OH} = -25\mu\text{A}$	2.7		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 0.36\text{mA}$		0.4	V
IP0-IP7, P8, P9, SKIP/P10	(Note 1)			
Logic High (COP404LS only)	$I_{OH} = -100\mu\text{A}$	2.4		V
Logic Low	$I_{OL} = 1.6\text{mA}$		0.4	V
Output Current Levels				
Output Sink Current				
SO and SK Outputs ( $I_{OL}$ )	$V_{CC} = 9.5\text{V}$ , $V_{OL} = 0.4\text{V}$ $V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	1.8 0.9		mA mA
$L_0$ - $L_7$ Outputs	$V_{CC} = 9.5\text{V}$ , $V_{OL} = 0.4\text{V}$ $V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.8 0.4		mA mA
$G_0$ - $G_3$ and $D_0$ - $D_3$ Outputs	$V_{CC} = 9.5\text{V}$ , $V_{OL} = 1.0\text{V}$ $V_{CC} = 4.5\text{V}$ , $V_{OL} = 1.0\text{V}$	30 15		mA mA
CKO (COP404LS)	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.2		mA

**COP404L****DC Electrical Characteristics** (continued)  $0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{\text{CC}} \leq 9.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Output Source Current:				
D <sub>0</sub> -D <sub>3</sub> , G <sub>0</sub> -G <sub>3</sub> Outputs (I <sub>OH</sub> )	V <sub>CC</sub> = 9.5V, V <sub>OH</sub> = 2.0V	-140	-800	μA
	V <sub>CC</sub> = 4.5V, V <sub>OH</sub> = 2.0V	-30	-250	μA
SO and SK Outputs (I <sub>OH</sub> )	V <sub>CC</sub> = 9.5V, V <sub>OH</sub> = 4.75V	-1.4		mA
	V <sub>CC</sub> = 4.5V, V <sub>OH</sub> = 1.0V	-1.2		mA
L <sub>0</sub> -L <sub>7</sub> Outputs	V <sub>CC</sub> = 9.5V, V <sub>OH</sub> = 2.0V	-3.0	-35	mA
	V <sub>CC</sub> = 6.0V, V <sub>OH</sub> = 2.0V	-3.0	-25	mA
Input Load Source Current (I <sub>IL</sub> )	V <sub>CC</sub> = 5.0V, V <sub>IL</sub> = 0V	-10	-140	μA
Total Sink Current Allowed				
All Outputs Combined			140	mA
D, G Ports			120	mA
L <sub>7</sub> -L <sub>4</sub>			4	mA
L <sub>3</sub> -L <sub>0</sub>			4	mA
All Other Pins			1.8	mA
Total Source Current Allowed				
All I/O Combined			120	mA
L <sub>7</sub> -L <sub>4</sub>			60	mA
L <sub>3</sub> -L <sub>0</sub>			60	mA
Each L Pin			30	mA
All Other Pins			1.5	mA

**COP304L****Absolute Maximum Ratings**

Voltage at Any Pin Relative to GND	-0.5V to +10V
Ambient Operating Temperature	-40°C to +85°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C
Power Dissipation	0.75 Watt at 25°C 0.25 Watt at 85°C
Total Source Current	120 mA
Total Sink Current	140 mA

*Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

**DC Electrical Characteristics**  $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 7.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Operating Voltage ( $V_{CC}$ )	(Note 2)	4.5	7.5	V
Power Supply Ripple	peak to peak		0.5	V
Operating Supply Current	all inputs and outputs open		21	mA
Input Voltage Levels				
CKI Input Levels				
Crystal Input				
Logic High ( $V_{IH}$ )		2.2		V
Logic Low ( $V_{IL}$ )		-0.3	0.3	V
RESET Input Levels	Schmitt Trigger Input			
Logic High		$0.7V_{CC}$		V
Logic Low		-0.3	0.4	V
IP0-IP7, SI Input Levels				
Logic High	$V_{CC} = 7.5\text{V}$	2.4		V
Logic High	$V_{CC} = 5\text{V} \pm 5\%$	2.2		V
Logic Low		-0.3	0.6	V
All Other Inputs				
Logic High	high trip level options selected	3.6		V
Logic Low		-0.3	1.2	V
Input Capacitance			7	pF
Output Voltage Levels				
LSTTL Operation	$V_{CC} = 5\text{V} \pm 5\%$			
Logic High ( $V_{OH}$ )	$I_{OH} = -20\mu\text{A}$	2.7		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 0.36\text{mA}$		0.4	V
IP0-IP7, P8, P9, SKIP/P10	$R_L = 5.6\text{k}\Omega$ (Note 1)			
Logic High	$I_{OH} = -100\mu\text{A}$	2.4		V
Logic Low	$I_{OL} = 1.6\text{mA}$		0.4	V
Output Current Levels				
Output Sink Current				
SO and SK Outputs ( $I_{OL}$ )	$V_{CC} = 7.5\text{V}$ , $V_{OL} = 0.4\text{V}$	1.4		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.8		mA
$L_0$ - $L_7$ Outputs	$V_{CC} = 7.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.6		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.4		mA
$G_0$ - $G_3$ and $D_0$ - $D_3$ Outputs	$V_{CC} = 7.5\text{V}$ , $V_{OL} = 1.0\text{V}$	24		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 1.0\text{V}$	14		mA
CKO (COP404LS)	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.2		mA

**COP304L****DC Electrical Characteristics** (continued)  $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{\text{CC}} \leq 7.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Output Source Current: D <sub>0</sub> -D <sub>3</sub> , G <sub>0</sub> -G <sub>3</sub> Outputs (I <sub>OH</sub> )	V <sub>CC</sub> = 7.5V, V <sub>OH</sub> = 2.0V V <sub>CC</sub> = 4.5V, V <sub>OH</sub> = 2.0V	-100 -28	-900 -350	μA μA
SO and SK Outputs (I <sub>OH</sub> )	V <sub>CC</sub> = 7.5V, V <sub>OH</sub> = 3.75V V <sub>CC</sub> = 4.5V, V <sub>OH</sub> = 1.0V	-0.85 -1.2		mA mA
L <sub>0</sub> -L <sub>7</sub> Outputs	V <sub>CC</sub> = 7.5V, V <sub>OH</sub> = 2.0V V <sub>CC</sub> = 6.0V, V <sub>OH</sub> = 2.0V	-2.7 -2.7	-54 -34	mA mA
Input Load Source Current (I <sub>IL</sub> )	V <sub>CC</sub> = 5.0V, V <sub>IL</sub> = 0V	-10	-200	μA
Total Sink Current Allowed				
All Outputs Combined			140	mA
D, G Ports			120	mA
L <sub>7</sub> -L <sub>4</sub>			4	mA
L <sub>3</sub> -L <sub>0</sub>			4	mA
All Other Pins			1.8	mA
Total Source Current Allowed				
All I/O Combined			120	mA
L <sub>7</sub> -L <sub>4</sub>			60	mA
L <sub>3</sub> -L <sub>0</sub>			60	mA
Each L Pin			30	mA
All Other Pins			1.5	mA

**COP404L/COP304L****AC Electrical Characteristics**COP404L:  $0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{\text{CC}} \leq 9.5\text{V}$  unless otherwise specified.COP304L:  $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $4.5\text{V} \leq 7.5\text{V}$  unless otherwise specified

Parameter	Conditions	Min.	Max.	Units
Instruction Cycle Time		15	40	μs
CKI				
Input Frequency f <sub>i</sub>	(÷32 mode)	0.8	2.1	MHz
Duty Cycle		30	60	%
Rise Time	f <sub>i</sub> = 2.097 MHz		120	ns
Fall Time			80	ns
INPUTS:				
SI, IP7-IP0				
t <sub>SETUP</sub>			2.0	μs
t <sub>HOLD</sub>			1.0	μs
IN <sub>3</sub> -IN <sub>0</sub> , G <sub>3</sub> -G <sub>0</sub> , L <sub>7</sub> -L <sub>0</sub>				
t <sub>SETUP</sub>			8.0	μs
t <sub>HOLD</sub>			1.3	μs
OUTPUT PROPAGATION DELAY	Test condition: C <sub>L</sub> = 50 pF, V <sub>OUT</sub> = 1.5V			
SO, SK Outputs	R <sub>L</sub> = 20 kΩ			
t <sub>pd1</sub> , t <sub>pd0</sub>			4.0	μs
D <sub>3</sub> -D <sub>0</sub> , G <sub>3</sub> -G <sub>0</sub> , L <sub>7</sub> -L <sub>0</sub>	R <sub>L</sub> = 20 kΩ			
t <sub>pd1</sub> , t <sub>pd0</sub>			5.6	μs
IP7-IP0, P8, P9, SKIP	R <sub>L</sub> = 5 kΩ			
t <sub>pd1</sub> , t <sub>pd0</sub>			7.2	μs
P10				
t <sub>pd1</sub> , t <sub>pd0</sub>	R <sub>L</sub> = 5 kΩ		6.0	μs

**Note 1:** Pull-up resistors required on COP404LP *only*; COP404LS has Push-Pull drivers on these outputs.**Note 2:** V<sub>CC</sub> voltage change must be less than 0.5V in a 1ms period to maintain proper operation.

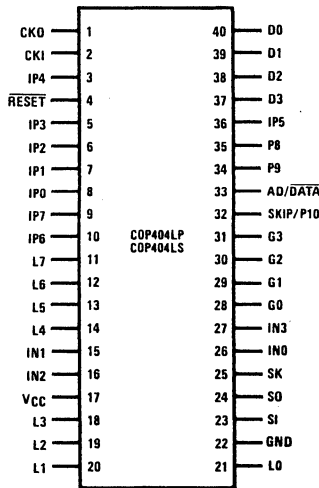
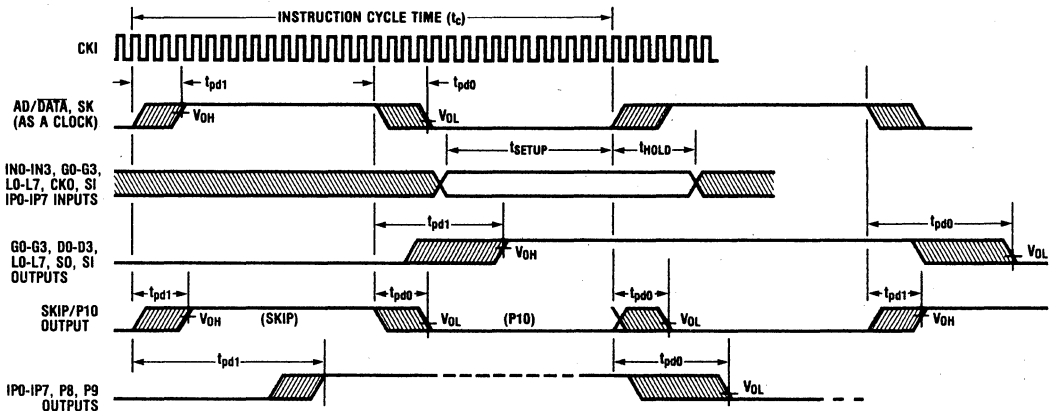


Figure 2. Connection Diagram

Order Number COP404L/N, COP304L/N  
NS Package N40A

Pin	Description	Pin	Description
L7-L0	8 bidirectional I/O ports with TRI-STATE®	CKI	System oscillator input
G3-G0	4 bidirectional I/O ports	CKO	General purpose input (COP404LP) System oscillator output (COP404LS)
D3-D0	4 general purpose outputs	RESET	System reset input
IN3-IN0	4 general purpose inputs	VCC	Power supply
SI	Serial input (or counter input)	GND	Ground
SO	Serial output (or general purpose output)	IP7-IP0	8 bidirectional ROM address and data ports
SK	Logic-controlled clock (or general purpose output)	P8, P9	2 ROM address outputs
AD/DATA	Address out/data in flag	SKIP/P10	instruction skip output and most significant ROM address bit output



## Functional Description

A block diagram of the COP404L is given in Figure 1. Data paths are illustrated in simplified form to depict how the various logic elements communicate with each other in implementing the instruction set of the device. Positive logic is used. When a bit is set, it is a logic "1" (greater than 2 volts). When a bit is reset, it is a logic "0" (less than 0.8 volts).

All functional references to the COP404L also apply to the COP304L.

### Program Memory

Program Memory consists of a 2048 byte external memory. As can be seen by an examination of the COP404L instruction set, these words may be program instructions, program data or ROM addressing data. Because of the special characteristics associated with the JP, JSRP, JID and LQID instructions, ROM must often be thought of as being organized into 32 pages of 64 words each.

ROM addressing is accomplished by a 11-bit PC register. Its binary value selects one of the 2048 8-bit words contained in ROM. A new address is loaded into the PC register during each instruction cycle. Unless the instruction is a transfer of control instruction, the PC register is loaded with the next sequential 11-bit binary count value. Three levels of subroutine nesting are implemented by the 11-bit subroutine saves registers, SA, SB, and SC, providing a last-in, first-out (LIFO) hardware subroutine stack.

ROM instruction words are fetched, decoded and executed by the Instruction Decode, Control and Skip Logic circuitry.

### Data Memory

Data memory consists of a 512-bit RAM, organized as 8 data registers of 16 4-bit digits. RAM addressing is implemented by a 7-bit B register whose upper 3 bits (Br) select 1 of 8 data registers and lower 4 bits (Bd) select 1 of 16 4-bit digits in the selected data register. While the 4-bit contents of the selected RAM digit (M) is usually loaded into or from, or exchanged with, the A register (accumulator), it may also be loaded into or from the Q latches or loaded from the L ports. RAM addressing may also be performed directly by the LDD and XAD instructions based upon the 7-bit contents of the operand of these instructions. The Bd register also serves as a source register for 4-bit data sent directly to the D outputs.

### Internal Logic

The 4-bit A register (accumulator) is the source and destination register for most I/O, arithmetic, logic and data memory access operations. It can also be used to load the Br and Bd portions of the B register, to load and input 4 bits of the 8-bit Q latch data, to input 4 bits of the 8-bit L I/O port data and to perform data exchanges with the SIO register.

A 4-bit adder performs the arithmetic and logic functions, storing its results in A. It also outputs a carry bit to the 1-bit C register, most often employed to indicate arithmetic overflow. The C register, in conjunction with the XAS instruction and the EN register, also serves to control the SK output. C can be outputted directly to SK or can enable SK to be a sync clock each instruction

cycle time. (See XAS instruction and EN register description, below.)

Four general-purpose inputs,  $IN_3$ – $IN_0$ , are provided.

The D register provides 4 general-purpose outputs and is used as the destination register for the 4-bit contents of Bd. The D outputs can be directly connected to the digits of a multiplexed LED display.

The G register contents are outputs to 4 general-purpose bidirectional I/O ports. G I/O ports can be directly connected to the digits of a multiplexed LED display.

The Q register is an internal, latched, 8-bit register, used to hold data loaded to or from M and A, as well as 8-bit data from ROM. Its contents are output to the L I/O ports when the L drivers are enabled under program control. (See LEI instruction.)

The 8 L drivers, when enabled, output the contents of latched Q data to the L I/O ports. Also, the contents of L may be read directly into A and M. L I/O ports can be directly connected to the segments of a multiplexed LED display (using the LED Direct Drive output configuration option) with Q data being outputted to the Sa–Sg and decimal point segments of the display.

The SIO register functions as a 4-bit serial-in/serial-out shift register or as a binary counter depending on the contents of the EN register. (See EN register description, below.) Its contents can be exchanged with A, allowing it to input or output a continuous serial data stream. SIO may also be used to provide additional parallel I/O by connecting SO to external serial-in/parallel-out shift registers.

The XAS instruction copies C into the SKL latch. In the counter mode, SK is the output of SKL; in the shift register mode, SK outputs SKL ANDed with the clock.

The EN register is an internal 4-bit register loaded under program control by the LEI instruction. The state of each bit of this register selects or deselected the particular feature associated with each bit of the EN register ( $EN_3$ – $EN_0$ ).

1. The least significant bit of the enable register,  $EN_0$ , selects the SIO register as either a 4-bit shift register or a 4-bit binary counter. With  $EN_0$  set, SIO is an asynchronous binary counter, *decrementing* its value by one upon each low-going pulse ("1" to "0") occurring on the SI input. Each pulse must be at least two instruction cycles wide. SK outputs the value of SKL. The SO output is equal to the value of  $EN_3$ . With  $EN_0$  reset, SIO is a serial shift register shifting left each instruction cycle time. The data present at SI goes into the least significant bit of SIO. SO can be enabled to output the most significant bit of SIO each cycle time. (See 4 below.) The SK output becomes a logic-controlled clock.
2. With  $EN_1$  set the  $IN_1$  input is enabled as an interrupt input. Immediately following an interrupt,  $EN_1$  is reset to disable further interrupts.
3. With  $EN_2$  set, the L drivers are enabled to output the data in Q to the L I/O ports. Resetting  $EN_2$  disables the L drivers, placing the L I/O ports in a high-impedance input state.

4. EN<sub>3</sub>, in conjunction with EN<sub>0</sub>, affects the SO output. With EN<sub>0</sub> set (binary counter option selected) SO will output the value loaded into EN<sub>3</sub>. With EN<sub>0</sub> reset (serial shift register option selected), setting EN<sub>3</sub> enables SO as the output of the SIO shift register, outputting serial shifted data each instruction time. Resetting EN<sub>3</sub> with the serial shift register option selected disables SO as the shift register output; data continues to be shifted through SIO and can be exchanged with A via an XAS instruction but SO remains reset to "0." The table below provides a summary of the modes associated with EN<sub>3</sub> and EN<sub>0</sub>.

**Interrupt**

The following features are associated with the IN<sub>1</sub> interrupt procedure and protocol and must be considered by the programmer when utilizing interrupts.

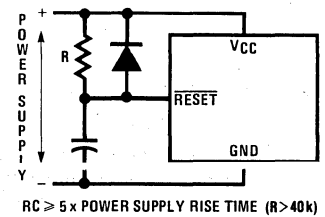
- a. The interrupt, once acknowledged as explained below, pushes the next sequential program counter address (PC + 1) onto the stack, pushing in turn the contents of the other subroutine-save registers to the next lower level (PC + 1 → SA → SB → SC). Any previous contents of SC are lost. The program counter is set to hex address 0FF (the last word of page 3) and EN<sub>1</sub> is reset.
- b. An interrupt will be acknowledged only after the following conditions are met:
  - 1. EN<sub>1</sub> has been set.
  - 2. A low-going pulse ("1" to "0") at least two instruction cycles wide occurs on the IN<sub>1</sub> input.
  - 3. A currently executing instruction has been completed.
  - 4. All successive transfer of control instructions and successive LBIs have been completed (e.g., if the main program is executing a JP instruction which transfers program control to another JP instruction, the interrupt will not be acknowledged until the second JP instruction has been executed).
- c. Upon acknowledgement of an interrupt, the skip logic status is saved and later restored upon popping of the stack. For example, if an interrupt occurs during the execution of ASC (Add with Carry, Skip on Carry) instruction which results in carry, the skip logic status

is saved and program control is transferred to the interrupt servicing routine at hex address 0FF. At the end of the interrupt routine, a RET instruction is executed to "pop" the stack and return program control to the instruction following the original ASC. At this time, the skip logic is enabled and skips this instruction because of the previous ASC carry. Subroutines and LQID instructions should not be nested within the interrupt service routine, since their popping the stack will enable any previously saved main program skips, interfering with the orderly execution of the interrupt routine.

- d. The first instruction of the interrupt routine at hex address 0FF must be a NOP.
- e. A LEI instruction can be put immediately before the RET to re-enable interrupts.

**Initialization**

The Reset Logic will initialize (clear) the device upon power-up if the power supply rise time is less than 1 ms and greater than 1 μs. If the power supply rise time is greater than 1 ms, the user must provide an external RC network and diode to the RESET pin as shown below. The RESET pin is configured as a Schmitt trigger input. If the RC network is not used, the RESET pin should be left open. Initialization will occur whenever a logic "0" is applied to the RESET input, provided it stays low for at least three instruction cycle times.



Enable Register Modes — Bits EN<sub>3</sub> and EN<sub>0</sub>

EN <sub>3</sub>	EN <sub>0</sub>	SIO	SI	SO	SK
0	0	Shift Register	Input to Shift Register	0	If SKL = 1, SK = CLOCK If SKL = 0, SK = 0
1	0	Shift Register	Input to Shift Register	Serial Out	If SKL = 1, SK = CLOCK If SKL = 0, SK = 0
0	1	Binary Counter	Input to Binary Counter	0	If SKL = 1, SK = 1 If SKL = 0, SK = 0
1	1	Binary Counter	Input to Binary Counter	1	If SKL = 1, SK = 1 If SKL = 0, SK = 0



Upon initialization, the PC register is cleared to 0 (ROM address 0) and the A, B, C, D, EN, and G registers are cleared. The SK output is enabled as a SYNC output, providing a pulse each instruction cycle time. *Data Memory (RAM) is not cleared upon initialization.* The first instruction at address 0 must be a CLRA.

### External Memory Interface

The COP404L is designed for use with an external Program Memory. This memory may be implemented using any devices having the following characteristics:

1. random addressing
2. TTL-compatible TRI-STATE® outputs
3. TTL-compatible inputs
4. access time = 5 $\mu$ s max.

Typically these requirements are met using bipolar or MOS PROMs.

During operation, the address of the next instruction is sent out on P10, P9, P8, and IP7 through IP0 during the time that AD/DATA is high (logic "1" = address mode). Address data on the IP lines is stored into an external latch on the high-to-low transition of the AD/DATA line; P9 and P8 are dedicated address outputs, and do not need to be latched. SKIP/P10 outputs address data when AD/DATA is low. When AD/DATA is low (logic "0" = data mode), the output of the memory is gated onto IP7 through IP0, forming the input bus. Note that the AD/DATA output has a period of one instruction time, a duty cycle of approximately 50%, and specifies whether the IP lines are used for address output or instruction input.

### Oscillator

Two basic clock oscillator configurations have been implemented, as shown in Figure 4.

- a. **Crystal Controlled Oscillator** (COP404LS only). CKI and CKO are connected to an external crystal. The instruction cycle time equals the crystal frequency divided by 32
- b. **External Oscillator** (COP404LP only). CKI is an external clock input signal. The external frequency is divided by 32 to give the instruction cycle time. CKO is used as a general purpose input.

### CKO as an Input

On the COP404LP, CKO has been configured as a general-purpose input. The logic level applied to CKO will be read into bit 2 of A (accumulator) upon execution of an INIL instruction.

### Input/Output Configurations

COP404L outputs have the following configurations, illustrated in figure 5:

- a. **Standard** — an enhancement mode device to ground in conjunction with a depletion-mode device to V<sub>CC</sub>, compatible with LSTTL and CMOS input requirements. (Used on D and G outputs.)

b. **Open-Drain** — an enhancement-mode device to ground only, allowing external pull-up as required by the user's application. (Used on IP, P and SKIP/P10 outputs on COP404LP only.)

c. **Push-Pull** — an enhancement-mode device to ground in conjunction with a depletion-mode device paralleled by an enhancement-mode device to V<sub>CC</sub>. This configuration has been provided to allow for fast rise and fall times when driving capacitive loads. (Used on SO and SK outputs on COP404LP and 404LS; also used on IP, P and SKIP/P10 outputs on COP404LS only.)

d. **LED Direct Drive** — an enhancement-mode device to ground and to V<sub>CC</sub>, meeting the typical current sourcing requirements of the segments of an LED display. The sourcing device is clamped to limit current flow. These devices may be turned off under program control (See Functional Description, EN Register), placing the outputs in a high-impedance state to provide required LED segment blanking for a multiplexed display. (Used on L outputs).

COP404L inputs have an on-chip depletion load device to V<sub>CC</sub>.

The above input and output configurations share common enhancement-mode and depletion-mode devices. Specifically, all configurations use one or more of six devices (numbered 1-6, respectively). Minimum and maximum current (I<sub>OUT</sub> and V<sub>OUT</sub>) curves are given in Figure 6 for each of these devices to allow the designer to effectively use these I/O configurations in designing a system.

An important point to remember is that even when the L drivers are disabled, the depletion load device will source a small amount of current (see Figure 6, device 2); however, when the L-lines are used as inputs, the disabled depletion device can *not* be relied on to source sufficient current to pull an input to a logic "1".

### COP404LP and COP404LS

Two versions of the basic COP404L have been implemented: the COP404LP, with open-drain memory interface drivers, is used only in the COP400-E04L Emulator Card; the COP404LS, with push-pull memory interface, is intended for use in small to medium volume production applications.

The COP404LP has an oscillator output option on CKO; the COP404LS has a general purpose input option.

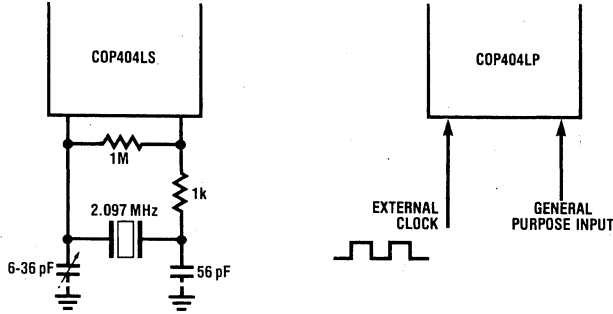
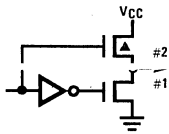
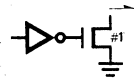


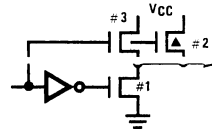
Figure 4. Oscillator



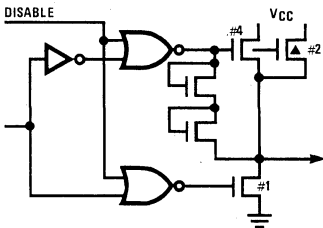
a. Standard Output



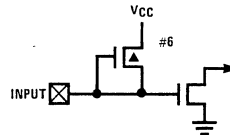
b. Open-Drain Output



c. Push-Pull Output



d. L Output (LED)



e. Input with Load

(▲ IS DEPLETION DEVICE)

Figure 5. Output Configurations

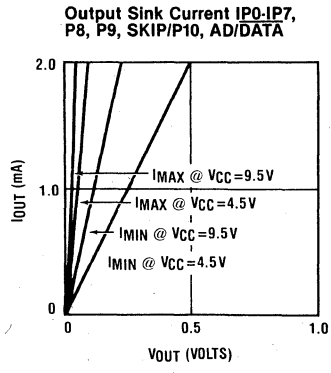
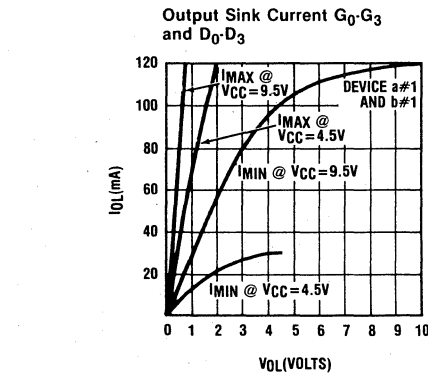
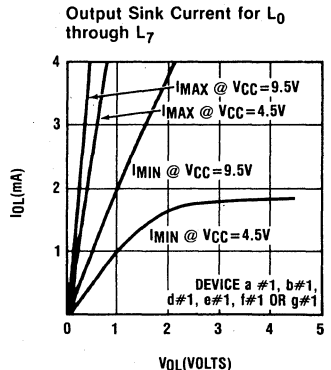
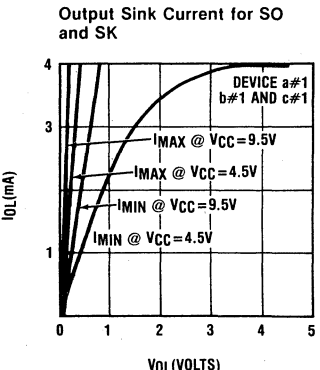
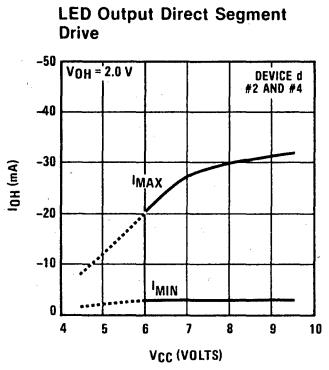
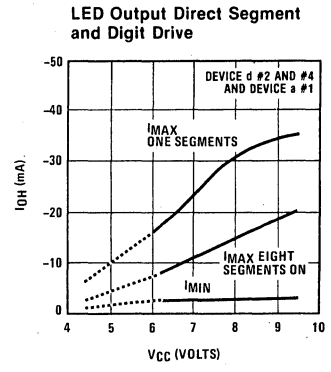
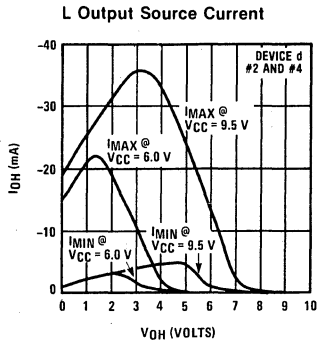
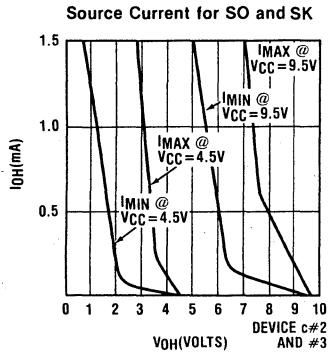
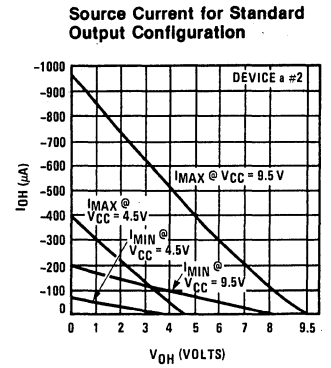
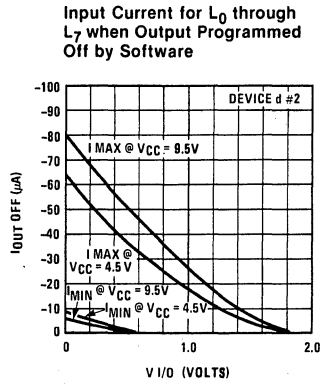
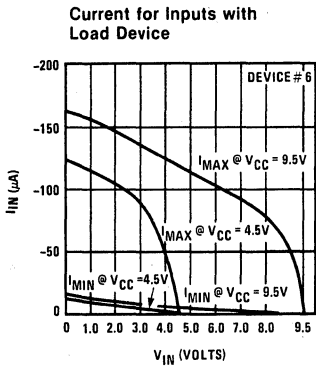


Figure 6a. COP404L I/O Characteristics

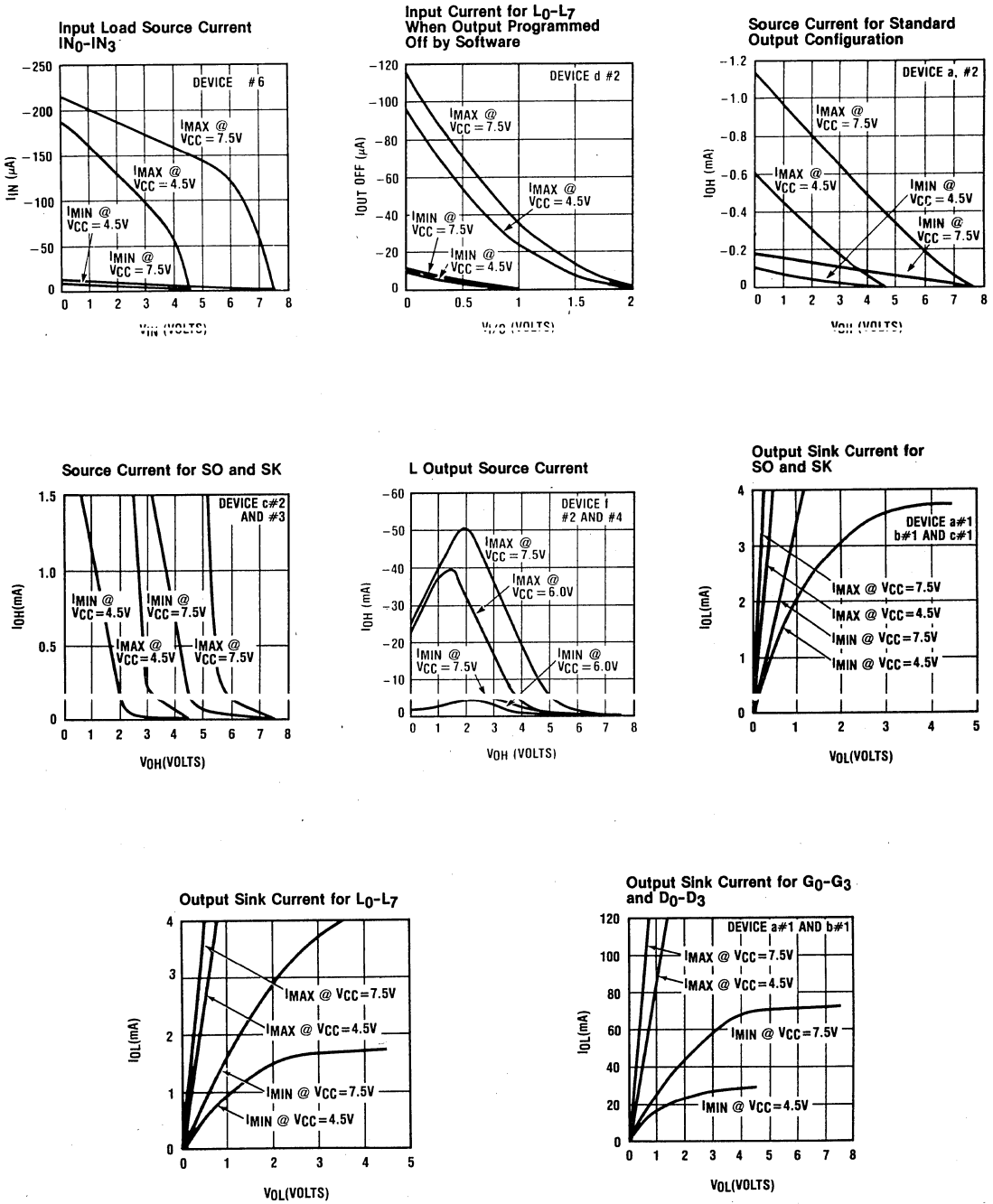


Figure 6b. COP304L I/O Characteristics

### COP404L/COP304L Instruction Set

Table 1 is a symbol table providing internal architecture, instruction operand and operational symbols used in the instruction set table.

Table 2 provides the mnemonic, operand, machine code, data flow, skip conditions, and description associated with each instruction in the COP404L/COP304L instruction set.

Table 2. COP404L/304L Instruction Set Table Symbols

Symbol	Definition	Symbol	Definition
<b>INTERNAL ARCHITECTURE SYMBOLS</b>		<b>INSTRUCTION OPERAND SYMBOLS</b>	
A	4-bit Accumulator	d	4-bit Operand Field, 0-15 binary (RAM Digit Select)
B	10-bit RAM Address Register	r	3-bit Operand Field, 0-7 binary (RAM Register Select)
Br	Upper 3 bits of B (register address)	a	11-bit Operand Field, 0-2047 binary (ROM Address)
Bd	Lower 4 bits of B (digit address)	y	4-bit Operand Field, 0-15 binary (Immediate Data)
C	1-bit Carry Register		RAM(s) Contents of RAM location addressed by s
D	4-bit Data Output Port		ROM(t) Contents of ROM location addressed by t
EN	4-bit Enable Register		
G	4-bit Register to latch data for G I/O Port		
IL	Two 1-bit latches associated with the IN <sub>3</sub> or IN <sub>0</sub> inputs		
IN	4-bit Input Port		
IP	8-bit bidirectional ROM address and Data Port		
L	8-bit TRI-STATE® I/O Port		
M	4-bit contents of RAM Memory pointed to by B Register		
P	3-bit ROM Address Register Port		
PC	11-bit ROM Address Register (program counter)		
Q	8-bit Register to latch data for L I/O Port		
SA	11-bit Subroutine Save Register A		
SB	11-bit Subroutine Save Register B		
SC	11-bit Subroutine Save Register C		
SIO	4-bit Shift Register and Counter		
SK	Logic-Controlled Clock Output		
		<b>OPERATIONAL SYMBOLS</b>	
		+	Plus
		-	Minus
		→	Replaces
		↔	Is exchanged with
		=	Is equal to
		$\bar{A}$	The one's complement of A
		⊕	Exclusive-OR
		:	Range of values

Table 2. COP404L/304L Instruction Set

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>ARITHMETIC INSTRUCTIONS</b>						
ASC		30	$[0011 0000]$	$A + C + RAM(B) \rightarrow A$ Carry $\rightarrow C$	Carry	Add with Carry, Skip on Carry
ADD		31	$[0011 0001]$	$A + RAM(B) \rightarrow A$	None	Add RAM to A
ADT		4A	$[0100 1010]$	$A + 10_{10} \rightarrow A$	None	Add Ten to A
AISC	y	5-	$[0101 y]$	$A + y \rightarrow A$	Carry	Add Immediate, Skip on Carry (y $\neq$ 0)
CASC		10	$[0001 0000]$ s	$\bar{A} + RAM(B) + C \rightarrow A$ Carry $\rightarrow C$	Carry	Complement and Add with Carry, Skip on Carry
CLRA		00	$[0000 0000]$	$0 \rightarrow A$	None	Clear A
COMP		40	$[0100 0000]$	$\bar{A} \rightarrow A$	None	One's complement of A to A
NOP		44	$[0100 0100]$	None	None	No Operation
RC		32	$[0011 0010]$	"0" $\rightarrow C$	None	Reset C
SC		22	$[0010 0010]$	"1" $\rightarrow C$	None	Set C
XOR		02	$[0000 0010]$	$A \oplus RAM(B) \rightarrow A$	None	Exclusive-OR RAM with A
<b>TRANSFER OF CONTROL INSTRUCTIONS</b>						
JID		FF	$[1111 1111]$	ROM (PC <sub>10:8</sub> , A, M) $\rightarrow$ PC <sub>7:0</sub>	None	Jump Indirect (Note 3)
JMP	a	8-	$[0110 0 a_{10:8}]$ a <sub>7:0</sub>	a $\rightarrow$ PC	None	Jump
JP	a	--	$[1 a_{6:0}]$ (pages 2,3 only)	a $\rightarrow$ PC <sub>6:0</sub>	None	Jump within Page (Note 4)
			or $[1 a_{6:0}]$ (all other pages)	a $\rightarrow$ PC <sub>6:0</sub>		
JSRP	a	--	$[10 a_{5:0}]$	PC + 1 $\rightarrow$ SA $\rightarrow$ SB $\rightarrow$ SC 00010 $\rightarrow$ PC <sub>10:6</sub> a $\rightarrow$ PC <sub>5:0</sub>	None	Jump to Subroutine Page (Note 5)
JSR	a	6-	$[0110 1 a_{10:8}]$	PC + 1 $\rightarrow$ SA $\rightarrow$ SB $\rightarrow$ SC	None	Jump to Subroutine
			a <sub>7:0</sub>	a $\rightarrow$ PC		
RET		48	$[0100 1000]$	SC $\rightarrow$ SB $\rightarrow$ SA $\rightarrow$ PC	None	Return from Subroutine
RETSK		49	$[0100 1001]$	SC $\rightarrow$ SB $\rightarrow$ SA $\rightarrow$ PC	Always Skip on Return	Return from Subroutine then Skip

Table 2. COP404L/304L Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>MEMORY REFERENCE INSTRUCTIONS</b>						
CAMQ		33	$\boxed{0011 0011}$	A → Q <sub>7:4</sub>	None	Copy A, RAM to Q
		3C	$\boxed{0011 1100}$	RAM(B) → Q <sub>3:0</sub>		
CQMA		33	$\boxed{0011 0011}$	Q <sub>7:4</sub> → RAM(B)	None	Copy Q to RAM, A
		2C	$\boxed{0010 1100}$	Q <sub>3:0</sub> → A		
LD	r	-5	$\boxed{00 r 0101}$ (r = 0:3)	RAM(B) → A Br ⊕ r → Br	None	Load RAM into A, Exclusive-OR Br with r
LDD	r,d	23	$\boxed{0010 0011}$	RAM(r,d) → A	None	Load A with RAM pointed to directly by r,d
		--	$\boxed{0 r d}$			
LQID		BF	$\boxed{1011 1111}$	ROM(PC <sub>10:8</sub> ,A,M) → Q SB → SC	None	Load Q Indirect (Note 3)
RMB	0	4C	$\boxed{0100 1100}$	0 → RAM(B) <sub>0</sub>	None	Reset RAM Bit
	1	45	$\boxed{0100 0101}$	0 → RAM(B) <sub>1</sub>		
	2	42	$\boxed{0100 0010}$	0 → RAM(B) <sub>2</sub>		
	3	43	$\boxed{0100 0011}$	0 → RAM(B) <sub>3</sub>		
SMB	0	4D	$\boxed{0100 1101}$	1 → RAM(B) <sub>0</sub>	None	Set RAM Bit
	1	47	$\boxed{0100 1101}$	1 → RAM(B) <sub>1</sub>		
	2	46	$\boxed{0100 0110}$	1 → RAM(B) <sub>2</sub>		
	3	4B	$\boxed{0100 1011}$	1 → RAM(B) <sub>3</sub>		
STII	y	7-	$\boxed{0111 y}$	y → RAM(B) Bd + 1 → Bd	None	Store Memory Immediate and Increment Bd
X	r	-6	$\boxed{00 r 0110}$ (r = 0:3)	RAM(B) ↔ A Br ⊕ r → Br	None	Exchange RAM with A, Exclusive-OR Br with r
XAD	r,d	23	$\boxed{0010 0011}$	RAM(r,d) ↔ A	None	Exchange A with RAM pointed to directly by r,d
		--	$\boxed{1 r d}$			
XDS	r	-7	$\boxed{00 r 0111}$ (r = 0:3)	RAM(B) ↔ A Bd - 1 → Bd Br ⊕ r → Br	Bd decrements past 0	Exchange RAM with A and Decrement Bd, Exclusive-OR Br with r
XIS	r	-4	$\boxed{00 r 0100}$ (r = 0:3)	RAM(B) ↔ A Bd + 1 → Bd Br ⊕ r → Br	Bd increments past 15	Exchange RAM with A and Increment Bd, Exclusive-OR Br with r
<b>REGISTER REFERENCE INSTRUCTIONS</b>						
CAB		50	$\boxed{0101 0000}$	A → Bd	None	Copy A to Bd
CBA		4E	$\boxed{0100 1110}$	Bd → A	None	Copy Bd to A
LBI	r,d	--	$\boxed{00 r (d-1)}$ (r = 0:3; d = 0, 9:15)	r,d → B	Skip until not a LBI	Load B Immediate with r,d (Note 6)
		33	$\boxed{0011 0011}$			
		--	$\boxed{1 r d}$ (any r, any d)			
LEI	y	33	$\boxed{0011 0001}$	y → EN	None	Load EN Immediate (Note 7)
		6-	$\boxed{0110 y}$			
XABR		12	$\boxed{0001 0010}$	A ↔ Br (0 → A <sub>3</sub> )	None	Exchange A with Br

Table 2. COP404L/304L Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>TEST INSTRUCTIONS</b>						
SKC		20	0010 0000		C = "1"	Skip if C is True
SKE		21	0010 0001		A = RAM(B)	Skip if A Equals RAM
SKGZ		33	0011 0011		G <sub>3:0</sub> = 0	Skip if G is Zero (all 4 bits)
		21	0010 0001			
SKGBZ		33	0011 0011	1st byte		Skip if G Bit is Zero
	0	01	0000 0001		G <sub>0</sub> = 0	
	1	11	0001 0001	2nd byte	G <sub>1</sub> = 0	
	2	03	0000 0011		G <sub>2</sub> = 0	
SKMBZ	3	13	0001 0011		G <sub>3</sub> = 0	
	0	01	0000 0001		RAM(B) <sub>0</sub> = 0	Skip if RAM Bit is Zero
	1	11	0001 0001		RAM(B) <sub>1</sub> = 0	
2	03	0000 0011		RAM(B) <sub>2</sub> = 0		
3	13	0001 0011		RAM(B) <sub>3</sub> = 0		
SKT		41	0100 0001		A time-base counter carry has occurred since last test	Skip on Timer (Note 2)
<b>INPUT/OUTPUT INSTRUCTIONS</b>						
ING		33	0011 0011	G → A	None	Input G Ports to A
		2A	0010 1010			
ININ		33	0011 0011	IN → A	None	Input IN Inputs to A
		28	0010 1000			
INIL		33	0011 0011	IL <sub>3</sub> , CKO, "0", IL <sub>0</sub> → A	None	Input IL Latches to A (Note 2)
		29	0010 1001			
INL		33	0011 0011	L <sub>3:1</sub> → RAM(B)	None	Input L Ports to RAM, A
		2E	0010 1110	L <sub>3:0</sub> → A		
OBD		33	0011 0011	Bd → D	None	Output Bd to D Outputs
		3E	0011 1110			
OGI	y	33	0011 0011	y → G	None	Output to G Ports Immediate
		5-	0101  y			
OMG		33	0011 0011	RAM(B) → G	None	Output RAM to G Ports
		3A	0011 1010			
XAS		4F	0100 1111	A ↔ SIO, C → SKL	None	Exchange A with SIO (Note 2)

**Note 1:** All subscripts for alphabetical symbols indicate bit numbers unless explicitly defined (e.g., Br and Bd are explicitly defined). Bits are numbered 0 to N where 0 signifies the least significant bit (low-order, right-most bit). For example, A<sub>3</sub> indicates the most significant (left-most) bit of the 4-bit A register.

**Note 2:** For additional information on the operation of the XAS, JID, LQID, INIL, and SKT instructions, see below.

**Note 3:** The JP instruction allows a jump, while in subroutine pages 2 or 3, to any ROM location within the two-page boundary of pages 2 or 3. The JP instruction, otherwise, permits a jump to a ROM location within the current 64-word page. JP may not jump to the last word of a page.

**Note 4:** A JSRP transfers program control to subroutine page 2 (0010 is loaded into the upper 4 bits of P). A JSRP may not be used when in pages 2 or 3. JSRP may not jump to the last word in page 2.

**Note 5:** LBI is a single-byte instruction if d = 0, 9, 10, 11, 12, 13, 14, or 15. The machine code for the lower 4 bits equals the binary value of the "d" data minus 7, e.g., to load the lower four bits of B (Bd) with the value 9 (1001<sub>2</sub>), the lower 4 bits of the LBI instruction equal 8 (1000<sub>2</sub>). To load 0, the lower 4 bits of the LBI instruction should equal 15 (1111<sub>2</sub>).

**Note 6:** Machine code for operand field y for LEI instruction should equal the binary value to be latched into EN, where a "1" or "0" in each bit of EN corresponds with the selection or deselection of a particular function associated with each bit. (See Functional Description, EN Register.)



The following information is provided to assist the user in understanding the operation of several unique instructions and to provide notes useful to programmers in writing COP404L programs.

### XAS Instruction

XAS (Exchange A with SIO) exchanges the 4-bit contents of the accumulator with the 4-bit contents of the SIO register. The contents of SIO will contain serial-in/serial-out shift register or binary counter data, depending on the value of the EN register. An XAS instruction will also affect the SK output. (See Functional Description, EN Register, above.) If SIO is selected as a shift register, an XAS instruction must be performed once every 4 instruction cycles to effect a continuous data stream.

### JID Instruction

JID (Jump Indirect) is an indirect addressing instruction, transferring program control to a new ROM location pointed to indirectly by A and M. It loads the lower 8 bits of the ROM address register PC with the contents of ROM addressed by the 11-bit word, PC<sub>10:8</sub>, A, M. PC<sub>10</sub>, PC<sub>9</sub> and PC<sub>8</sub> are not affected by this instruction.

Note that JID requires 2 instruction cycles to execute.

### INIL Instruction

INIL (Input IL Latches to A) inputs 2 latches, IL<sub>3</sub> and IL<sub>0</sub> (see figure 7) and CKO into A. The IL<sub>3</sub> and IL<sub>0</sub> latches are set if a low-going pulse ("1" to "0") has occurred on the IN<sub>3</sub> and IN<sub>0</sub> inputs since the last INIL instruction, provided the input pulse stays low for at least two instruction times. Execution of an INIL inputs IL<sub>3</sub> and IL<sub>0</sub> into A3 and A0 respectively, and resets these latches to allow them to respond to subsequent low-going pulses on the IN<sub>3</sub> and IN<sub>0</sub> lines. INIL will input the state of CKO into A2 on the COP404LP ("1" into A2 for the COP404LS). A "0" is always placed in A1 upon the execution of an INIL. The general purpose inputs IN<sub>3</sub>-IN<sub>0</sub> are input to A upon execution of an ININ instruction. (See table 2, ININ instruction.) INIL is useful in recognizing pulses of short duration or pulses which occur too often to be read conveniently by an ININ instruction.

Note: IL latches are not cleared on reset.

### LQID Instruction

LQID (Load Q Indirect) loads the 8-bit Q register with the contents of ROM pointed to by the 11-bit word PC<sub>10</sub>, PC<sub>9</sub>, PC<sub>8</sub>, A, M. LQID can be used for table lookup or code conversion such as BCD to seven-segment. The LQID instruction "pushes" the stack (PC + 1 → SA → SB → SC) and replaces the least significant 8 bits of PC as follows: A → PC<sub>7:4</sub>, RAM(B) → PC<sub>3:0</sub>, leaving PC<sub>10</sub>, PC<sub>9</sub> and PC<sub>8</sub> unchanged. The ROM data pointed to by the new address is fetched and loaded into the Q latches. Next, the stack is "popped" (SC → SB → SA → PC), restoring the saved value of PC to continue sequential program execution. Since LQID pushes SB → SC, the previous contents of SC are lost. Also, when LQID pops the stack, the previously pushed contents of SB are left in SC. The net result is that the contents of SB are placed in SC (SB → SC). Note that LQID takes two instruction cycle times to execute.

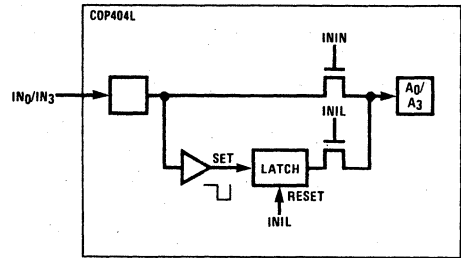


Figure 7. INIL Hardware Implementation

### SKT Instruction

The SKT (Skip On Timer) instruction tests the state of an internal 10-bit time-base counter. This counter divides the instruction cycle clock frequency by 1024 and provides a latched indication of counter overflow. The SKT instruction tests this latch, executing the next program instruction if the latch is not set. If the latch has been set since the previous test, the next program instruction is skipped and the latch is reset. The features associated with this instruction, therefore, allow the COP404L to generate its own time-base for real-time processing rather than relying on an external input signal.

For example, using a 2.097 MHz oscillator as the time-base to the clock generator, the instruction cycle clock frequency will be 65kHz (crystal frequency ÷ 32) and the binary counter output pulse frequency will be 64Hz. For time-of-day or similar real-time processing, the SKT instruction can call a routine which increments a "seconds" counter every 64 ticks.

### Instruction Set Notes

- The first word of a COP404L program (ROM address 0) must be a CLRA (Clear A) instruction.
- Although skipped instructions are not executed, one instruction cycle time is devoted to skipping each byte of the skipped instruction. Thus all program paths except JID and LQID take the same number of cycle times whether instructions are skipped or executed. JID and LQID instructions take 2 cycles if executed and 1 cycle if skipped.
- The ROM is organized into 32 pages of 64 words each. The Program Counter is an 11-bit binary counter, and will count through page boundaries. If a JP, JSRP, JID or LQID instruction is located in the last word of a page, the instruction operates as if it were in the next page. For example: a JP located in the last word of a page will jump to a location in the next page. Also, a LQID or JID located in the last word of page 3, 7, 11, 15, 19, 23 or 27 will access data in the next group of four pages.

# Typical Applications

## PROM-Based System

The COP404L may be used to exactly emulate the COP444L. Figure 8 shows the interconnect to implement a COP444L hardware emulation. This connection uses a MM2716 EPROM as external memory. Other memory can be used such as bipolar PROM or RAM.

Pins IP7-IP0 are bidirectional inputs and outputs. When the AD/DATA clocking output turns on, the EPROM drivers are disabled and IP7-IP0 output addresses. The 8-bit latch (MM74C373) latches the addresses to drive the memory.

When AD/DATA turns off, the EPROM is enabled and the IP7-IP0 pins will input the memory data. P8, P9 and SKIP/P10 output the most significant address bits to the memory. (SKIP output may be used for program debug if needed.)

The other 28 pins of the COP404L may be configured exactly the same as a COP444L. The COP404L V<sub>CC</sub> can vary from 4.5V to 9.5V. However, 5 volts is used for the memory.

For In-Circuit emulation, see also COP404LR.

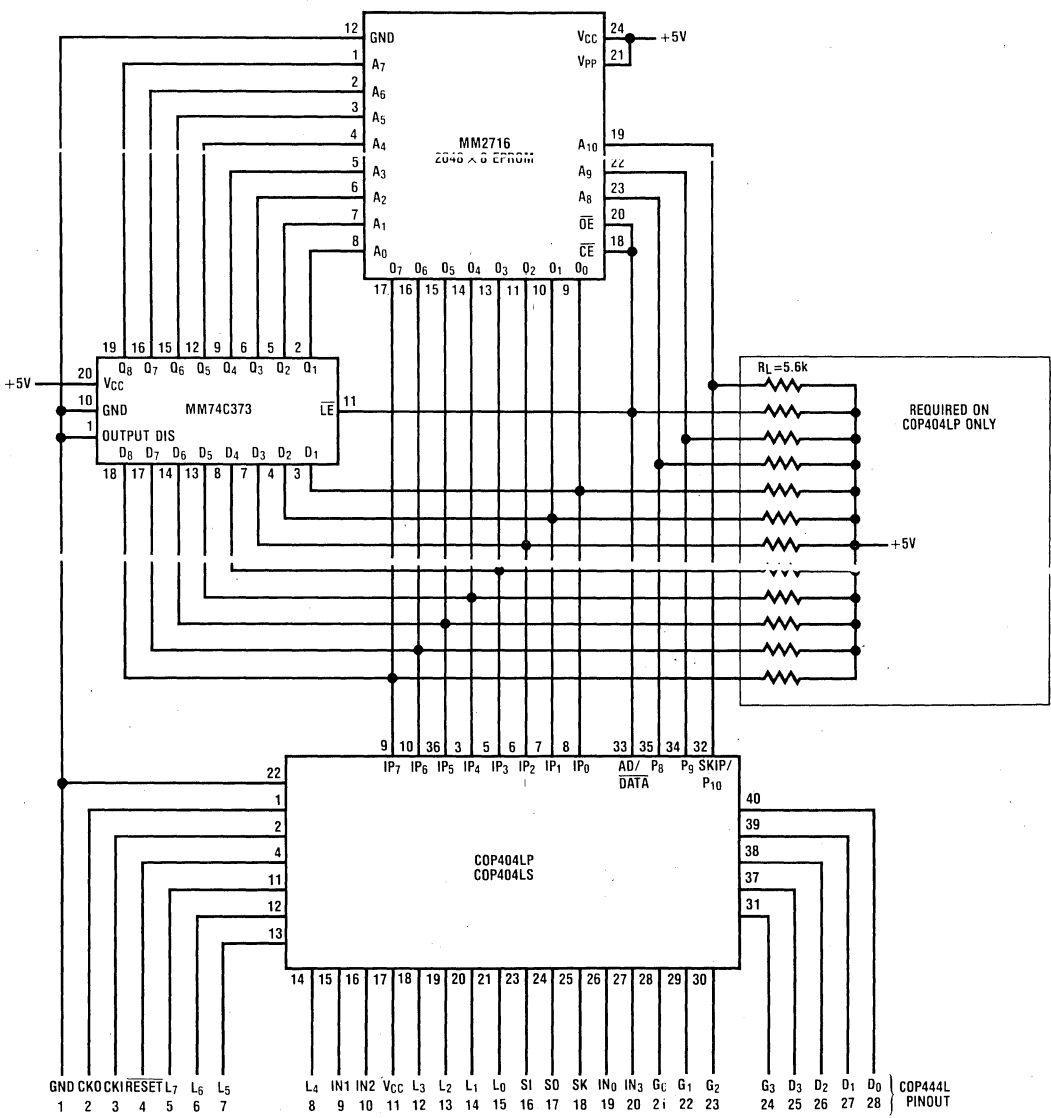


Figure 8. COP404L System Diagram

## COP404L Mask Options

The following COP444L options have been implemented on the basic versions of the COP404L:

Option Value	Comment	Option Value	Comment
Option 1 = 0	Ground, no option available	Option 17 = 2	SO has push-pull output
Option 2 = 0 (404LS)	CKO is clock generator output to crystal/resonator	Option 18 = 2	SK has push-pull output
= 2 (404LP)	CKO is general purpose input with load device to V <sub>CC</sub>	Option 19 = 0	IN0 has load device to V <sub>CC</sub>
Option 3 = 0	CKI is oscillator input (divide by 32)	Option 20 = 0	IN3 has load device to V <sub>CC</sub>
Option 4 = 0	RESET pin has load device to V <sub>CC</sub>	Option 21 = 0	G <sub>0</sub> }
Option 5 = 2	L <sub>7</sub> }	Option 22 = 0	G <sub>1</sub> } have very high current
Option 6 = 2	L <sub>6</sub> } have LED direct-drive	Option 23 = 0	G <sub>2</sub> } standard output
Option 7 = 2	L <sub>5</sub> } output	Option 24 = 0	G <sub>3</sub> }
Option 8 = 2	L <sub>4</sub> }	Option 25 = 0	D <sub>3</sub> }
Option 9 = 0	IN1 has load device to V <sub>CC</sub>	Option 26 = 0	D <sub>2</sub> } have very high current
Option 10 = 0	IN2 has load device to V <sub>CC</sub>	Option 27 = 0	D <sub>1</sub> } standard output
Option 11 = 1	V <sub>CC</sub> 4.5 to 9.5V operation	Option 28 = 0	D <sub>0</sub> }
Option 12 = 2	L <sub>3</sub> }	Option 29 = 1	L }
Option 13 = 2	L <sub>2</sub> } have LED direct-drive	Option 30 = 1	IN } have higher voltage
Option 14 = 2	L <sub>1</sub> } output	Option 31 = 1	G } input levels
Option 15 = 2	L <sub>0</sub> }	Option 32 = 0	SI has standard input level
Option 16 = 0	SI has load to V <sub>CC</sub>	Option 33 = 0	RESET has Schmitt trigger input
		Option 34 = 0	CKO has standard input levels
		Option 35 = N/A	40-pin package

# COP2404/COP2304 ROMless Dual CPU Microcontrollers

## General Description

The COP2404/COP2304 ROMless Dual CPU Microcontrollers are members of the COPS™ family, fabricated using N-channel, silicon gate MOS technology. Each microcontroller contains two identical CPUs with all system timing, internal logic, RAM and I/O necessary to implement dedicated control functions in a variety of applications, and are identical to COP2440/COP2340 devices, except that the ROM has been removed; pins have been added to output the ROM address and to input ROM data. In a system, the COP2404 will perform exactly as the COP2440; this important benefit facilitates development and debug of a COP2440 program prior to masking the final part. Features include single supply operation, various output configurations, and an instruction set, internal architecture, and I/O scheme designed to facilitate keyboard input, display output and data manipulation. Standard test procedures and reliable high-density fabrication techniques provide the medium to large volume customers with a dual CPU microcontroller at a low end-product cost. COP2304 is an exact functional equivalent version of COP2404 with an extended temperature range (-40°C to +85°C).

These microcontrollers are appropriate choices in many demanding control environments, especially those with human interface. Further, the high throughput and MICROBUS™ I/O facilitate numerous machine interface applications. The two CPUs provide on one chip the ability to handle two simultaneous but totally independent real time events.

TRI-STATE is a registered trademark of National Semiconductor Corp.  
COPS, MICROBUS, and MICROWIRE are trademarks of National Semiconductor Corp.

## Features

- Exact circuit equivalent of COP2440
- Standard 48-pin dual-in-line package
- Interfaces with standard PROM or ROM
- Two independent processors
- Dual CPU simplifies task partitioning — easy to program
- Enhanced, more powerful instruction set
- 160 × 4 RAM, addresses up to 2k × 8 ROM
- MICROBUS compatible
- Zero-crossing detect circuitry
- True multi-vectored interrupt from 4 selectable sources (plus restart)
- Four-level subroutine stack for each processor (in RAM)
- 4μs execution time per processor (non-overlapping)
- Single supply operation (4.5V–6.3V)
- Programmable time-base counter for real-time processing
- Internal binary counter/register with MICROWIRE™ compatible serial I/O
- General purpose and TRI-STATE® outputs
- TTL/CMOS compatible in and out
- Software/hardware compatible with other members of COP400 family
- Extended temperature range device COP2304 (-40°C to +85°C)
- Compatible single-processor device available

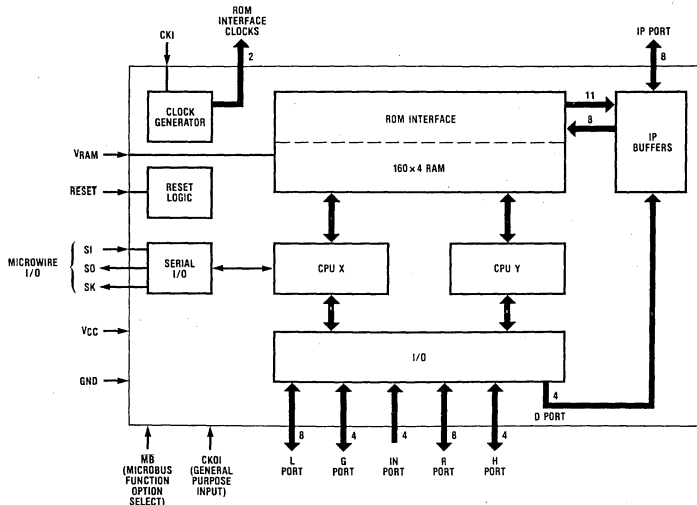


Figure 1. COP2404 Block Diagram

## COP2404

### Absolute Maximum Ratings

Voltage at Zero-Crossing Detect Pin Relative to GND	-1.2V to +15V
Voltage at Any Other Pin Relative to GND	-0.5V to +7V
Ambient Operating Temperature	0°C to +70°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C
Power Dissipation	0.75 Watt at 25°C 0.4 Watt at 70°C
Total Source Current	150 mA
Total Sink Current	90 mA

*Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

### DC Electrical Characteristics

0°C ≤ T<sub>A</sub> ≤ +70°C, 4.5V ≤ V<sub>CC</sub> ≤ 6.3V unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Operating Voltage (V <sub>CC</sub> )	Note 3	4.5	6.3	V
Power Supply Ripple	(peak to peak)		0.4	V
Operating Supply Current	(All inputs and outputs open) T <sub>A</sub> = 0°C T <sub>A</sub> = 25°C T <sub>A</sub> = 70°C		44 37 30	mA mA mA
V <sub>R</sub> RAM Power Supply Current	V <sub>R</sub> = 3.3V		3	mA
Input Voltage Levels				
CKI Input Levels (±16)				
Logic High (V <sub>IH</sub> )	V <sub>CC</sub> = Max.	2.5		V
Logic High (V <sub>IH</sub> )	V <sub>CC</sub> = 5V ± 5%	2.0		V
Logic Low (V <sub>IL</sub> )		-0.3	0.4	V
RESET Input Levels	(Schmitt Trigger Input)			
Logic High		0.7V <sub>CC</sub>		V
Logic Low		-0.3	0.6	V
Zero-Crossing Detect Input (IN <sub>1</sub> )	Zero-Crossing Interrupt Input; INIL Instruction			
Trip Point		-0.15	0.15	V
Logic High (V <sub>IH</sub> ) Limit			12	V
Logic Low (V <sub>IL</sub> ) Limit		-0.8		V
IN <sub>1</sub>				
Logic High	Interrupt Input; ININ Instruction;	3.0		V
Logic Low	MICROBUS™ Input	-0.3	0.8	V
All Other Inputs				
Logic High	V <sub>CC</sub> = Max.	2.5		V
Logic High	V <sub>CC</sub> = 5V ± 5%	2.0		V
Logic Low		-0.3	0.8	V
IN <sub>1</sub> Input Resistance to Ground	V <sub>IH</sub> = 1.0V	1.5	4.6	kΩ
Input Load Source Current	V <sub>IH</sub> = 2.0V, V <sub>CC</sub> = 4.5V	14	230	μA
Input Capacitance			7.0	pF
Hi-Z Input Leakage		-1.0	+1.0	μA

# COP2404

## DC Electrical Characteristics (Cont'd)

Parameter	Conditions	Min.	Max.	Units
Output Voltage Levels				
Standard Output				
TTL Operation				
Logic High ( $V_{OH}$ )	$I_{OH} = -100 \mu A$	2.4		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 1.6 \text{ mA}$		0.4	V
CMOS Operation				
Logic High ( $V_{OH}$ )	$I_{OH} = -10 \mu A$	$V_{CC} - 0.4$		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 10 \mu A$		0.2	V
TRI-STATE® Output				
TTL Operation				
Logic High ( $V_{OH}$ )	$I_{OH} = -100 \mu A$	2.4		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 1.6 \text{ mA}$		0.4	V
CMOS Operation	$33 \text{ k}\Omega \geq R_L \geq 4.7 \text{ k}\Omega$			
Logic High ( $V_{OH}$ )	$I_{OH} = -10 \mu A$	$V_{CC} - 0.5$		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 1.6 \text{ mA}$		0.4	V
Output Current Levels				
Standard Output Source Current	$V_{CC} = 4.5 \text{ V}, V_{OH} = 2.4 \text{ V}$	-100	-650	$\mu A$
TRI-STATE Output Leakage Current		-2.5	+2.5	$\mu A$
Total Sink Current Allowed				
All I/O Combined			90	mA
Each L, R Port			20	mA
Each D, G, H Port			10	mA
SO, SK			2.5	mA
IP			1.8	mA
Total Source Current Allowed	Note 4			
All I/O Combined			150	mA
L Port			120	mA
L <sub>7</sub> -L <sub>4</sub>			70	mA
L <sub>3</sub> -L <sub>0</sub>			70	mA
Each L Pin			23	mA
All Other Output Pins			1.6	mA

**COP2304****Absolute Maximum Ratings**

Voltage at Zero-Crossing Detect Pin	
Relative to GND	-1.2V to +15V
Voltage at Any Other Pin Relative to GND	-0.5V to +7V
Ambient Operating Temperature	-40°C to +85°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C
Power Dissipation	0.75 Watt at 25°C 0.25 Watt at 85°C
Total Source Current	150 mA
Total Sink Current	90 mA

*Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

**DC Electrical Characteristics**  $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 5.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Operating Voltage ( $V_{CC}$ )	Note 3	4.5	5.5	V
Power Supply Ripple	(peak to peak)		0.4	V
Operating Supply Current	(All inputs and outputs open)			
	$T_A = -40^{\circ}\text{C}$		57	mA
	$T_A = 25^{\circ}\text{C}$		37	mA
	$T_A = 85^{\circ}\text{C}$		29	mA
$V_R$ RAM Power Supply Current	$V_R = 3.3\text{V}$		4	mA
Input Voltage Levels				
CKI Input Levels ( $\pm 16$ )				
Logic High ( $V_{IH}$ )		2.2		V
Logic Low ( $V_{IL}$ )		-0.3	0.3	V
$\overline{\text{RESET}}$ Input Levels	(Schmitt Trigger Input)			
Logic High		$0.7V_{CC}$		V
Logic Low		-0.3	0.4	V
Zero-Crossing Detect Input ( $\text{IN}_1$ )	Zero-Crossing Interrupt Input; INIL Instruction			
Trip Point		-0.15	0.15	V
Logic High ( $V_{IH}$ ) Limit			12	V
Logic Low ( $V_{IL}$ ) Limit		-0.8		V
$\text{IN}_1$				
Logic High	Interrupt Input; ININ Instruction;	3.3		V
Logic Low	MICROBUS™ Input	-0.3	0.6	V
All Other Inputs				
Logic High		2.2		V
Logic Low		-0.3	0.6	V
$\text{IN}_1$ Input Resistance to Ground	$V_{IH} = 1.0\text{V}$	1.4	4.6	k $\Omega$
Input Load Source Current	$V_{IH} = 2.0\text{V}$ , $V_{CC} = 4.5\text{V}$	14	230	$\mu\text{A}$
Input Capacitance			7.0	pF
Hi-Z Input Leakage		-2.0	+2.0	$\mu\text{A}$

## COP2304

## DC Electrical Characteristics (Cont'd)

Parameter	Conditions	Min.	Max.	Units
Output Voltage Levels				
Standard Output				
TTL Operation				
Logic High ( $V_{OH}$ )	$I_{OH} = -100\mu A$	2.4		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 1.6\text{ mA}$		0.4	V
CMOS Operation				
Logic High ( $V_{OH}$ )	$I_{OH} = -10\mu A$	$V_{CC} - 0.5$		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 10\mu A$		0.2	V
TRI-STATE® Output				
TTL Operation				
Logic High ( $V_{OH}$ )	$I_{OH} = -100\mu A$	2.2		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 1.6\text{ mA}$		0.4	V
CMOS Operation	$33\text{ k}\Omega \geq R_L \geq 4.7\text{ k}\Omega$			
Logic High ( $V_{OH}$ )	$I_{OH} = -10\mu A$	$V_{CC} - 0.7$		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 1.6\text{ mA}$		0.4	V
Output Current Levels				
Standard Output Source Current	$V_{CC} = 4.5\text{ V}, V_{OH} = 2.4\text{ V}$	-100	-800	$\mu A$
TRI-STATE Output Leakage Current		-5.0	+5.0	$\mu A$
Total Sink Current Allowed				
All I/O Combined			75	mA
Each L, R Port			20	mA
Each D, G, H Port			10	mA
SO, SK			2.5	mA
IP			1.8	mA
Total Source Current Allowed	Note 4			
All I/O Combined			150	mA
L Port			120	mA
L <sub>7</sub> -L <sub>4</sub>			70	mA
L <sub>3</sub> -L <sub>0</sub>			70	mA
Each L Pin			23	mA
All Other Output Pins			1.6	mA



## AC Electrical Characteristics

COP2404:  $0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 6.3\text{V}$  unless otherwise noted.

COP2304:  $-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 5.5\text{V}$  unless otherwise noted.

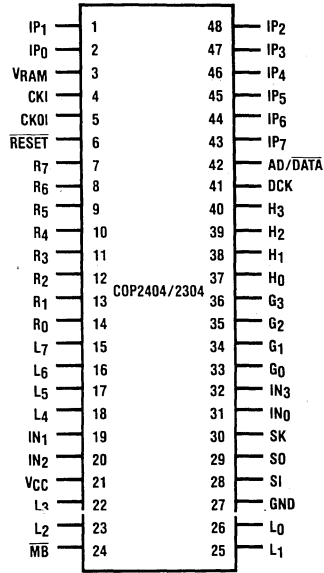
Parameter	Conditions	Min.	Max.	Units
Instruction Execution Time — $t_E$	Each Processor (Figure 3)	4.0	10	$\mu\text{s}$
CKI Frequency	$\div 16$ mode	1.6	4.0	MHz
Duty Cycle (Note 1)	$f_i = 4\text{ MHz}$	30	60	%
Rise Time	$f_i = 4\text{ MHz}$		60	ns
Fall Time	$f_i = 4\text{ MHz}$		40	ns
<b>INPUTS: (Figure 3)</b>				
SI				
$t_{\text{SETUP}}$		0.3		$\mu\text{s}$
$t_{\text{HOLD}}$		300		ns
IP				
$t_{\text{SETUP}}$		0.25		$\mu\text{s}$
$t_{\text{HOLD}}$		250		ns
$t_{\text{HOLD}}$	From $\overline{\text{AD}}/\overline{\text{DATA}}$ rising edge	0		ns
All Other Inputs				
$t_{\text{SETUP}}$		1.7		$\mu\text{s}$
$t_{\text{HOLD}}$		300		ns
<b>OUTPUT PROPAGATION DELAY</b>				
IP	Test Condition: $C_L = 50\text{ pF}$ , $V_{\text{OUT}} = 1.5\text{V}$			
$t_{\text{pd1A}}$ , $t_{\text{pd0A}}$			1.94	$\mu\text{s}$
$t_{\text{pd1B}}$ , $t_{\text{pd0B}}$			0.94	$\mu\text{s}$
DCK				
$t_{\text{pd1}}$ , $t_{\text{pd0}}$			375	ns
$\overline{\text{AD}}/\overline{\text{DATA}}$				
$t_{\text{pd1}}$ , $t_{\text{pd0}}$			300	ns
SO, SK				
$t_{\text{pd1}}$ , $t_{\text{pd0}}$	$R_L = 2.4\text{ k}\Omega$		1.0	$\mu\text{s}$
All Other Outputs	$R_L = 5.0\text{ k}\Omega$		1.4	$\mu\text{s}$
<b>MICROBUS<sup>TM</sup> TIMING</b>				
Read Operation		$C_L = 100\text{ pF}$ , $V_{CC} = 5\text{V} \pm 5\%$ TRI-STATE <sup>®</sup> outputs		
Chip Select Stable Before $\overline{\text{RD}}$ — $t_{\text{CSR}}$		65		ns
Chip Select Hold Time for $\overline{\text{RD}}$ — $t_{\text{RCS}}$		20		ns
$\overline{\text{RD}}$ Pulse Width — $t_{\text{RR}}$		400		ns
Data Delay from $\overline{\text{RD}}$ — $t_{\text{RD}}$			375	ns
$\overline{\text{RD}}$ to Data Floating — $t_{\text{DF}}$			250	ns
Write Operation				
Chip Select Stable Before $\overline{\text{WR}}$ — $t_{\text{CSW}}$		65		ns
Chip Select Hold Time for $\overline{\text{WR}}$ — $t_{\text{WCS}}$		20		ns
$\overline{\text{WR}}$ Pulse Width — $t_{\text{WW}}$		400		ns
Data Set-Up Time for $\overline{\text{WR}}$ — $t_{\text{DW}}$		320		ns
Data Hold Time for $\overline{\text{WR}}$ — $t_{\text{WD}}$		100		ns
INTR Transition Time from $\overline{\text{WR}}$ — $t_{\text{WI}}$			700	ns

**Note 1:** Duty Cycle =  $t_{\text{WI}} / (t_{\text{WI}} + t_{\text{WO}})$ .

**Note 2:** See Figure for additional I/O Characteristics.

**Note 3:**  $V_{CC}$  voltage change must be less than 0.5V in a 1 ms period to maintain proper operation.

**Note 4:** Exercise great care not to exceed maximum device power dissipation limits when direct-driving LEDs (or sourcing similar loads) at high temperature.



Order Number COP2404N, COP2304N  
NS Package N40A

Figure 2. Connection Diagram

Pin	Description
L <sub>7</sub> -L <sub>0</sub>	8-bit bidirectional TRI-STATE® I/O port
G <sub>3</sub> -G <sub>0</sub>	4-bit bidirectional I/O port
IN <sub>3</sub> -IN <sub>0</sub>	4-bit general purpose input port
H <sub>3</sub> -H <sub>0</sub>	4-bit bidirectional I/O port
R <sub>7</sub> -R <sub>0</sub>	8-bit bidirectional TRI-STATE I/O port
SI	Serial input
SO	Serial output (or general purpose output)
SK	Logic-controlled clock (or general purpose output)
CKI	System oscillator input
CKO1	General purpose input
VRAM	Power supply for first 4 registers of RAM
$\overline{MB}$	MICROBUS™ function select
DCK	Clock output to latch D outputs and high order address bits
AD/DATA	Address out/data in flag
IP <sub>7</sub> -IP <sub>0</sub>	8-bit directional port for ROM address, ROM data and D outputs
$\overline{RESET}$	System reset input
V <sub>CC</sub>	Power supply
GND	Ground

### Timing Diagram

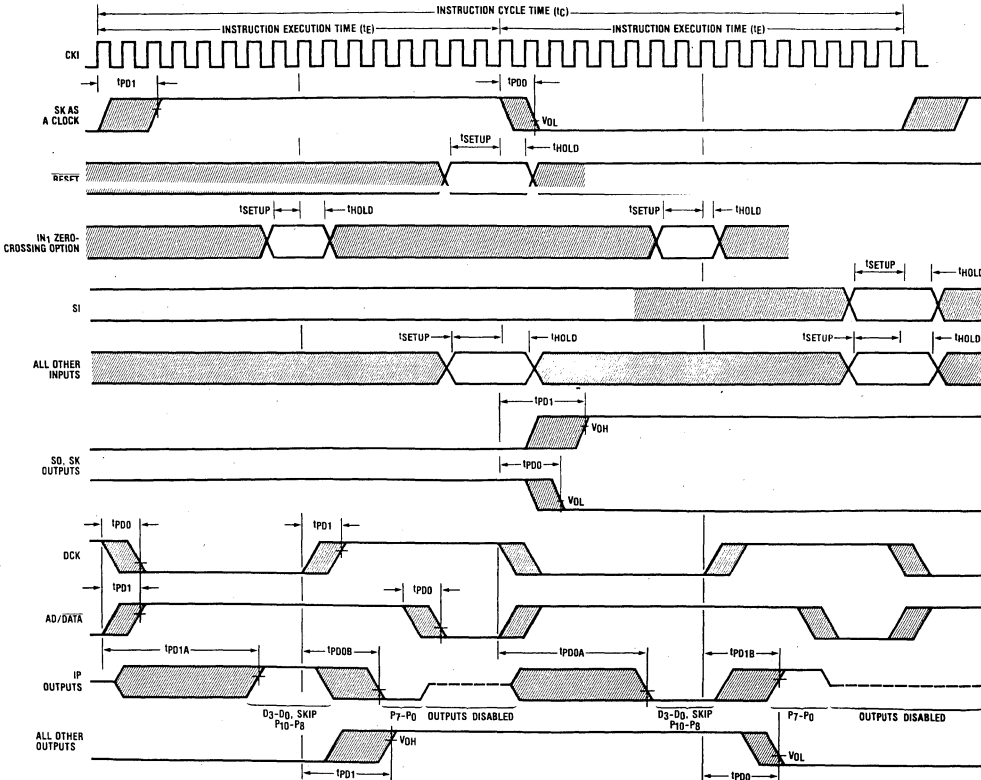


Figure 3. Input/Output Timing Diagrams (+16 Mode)

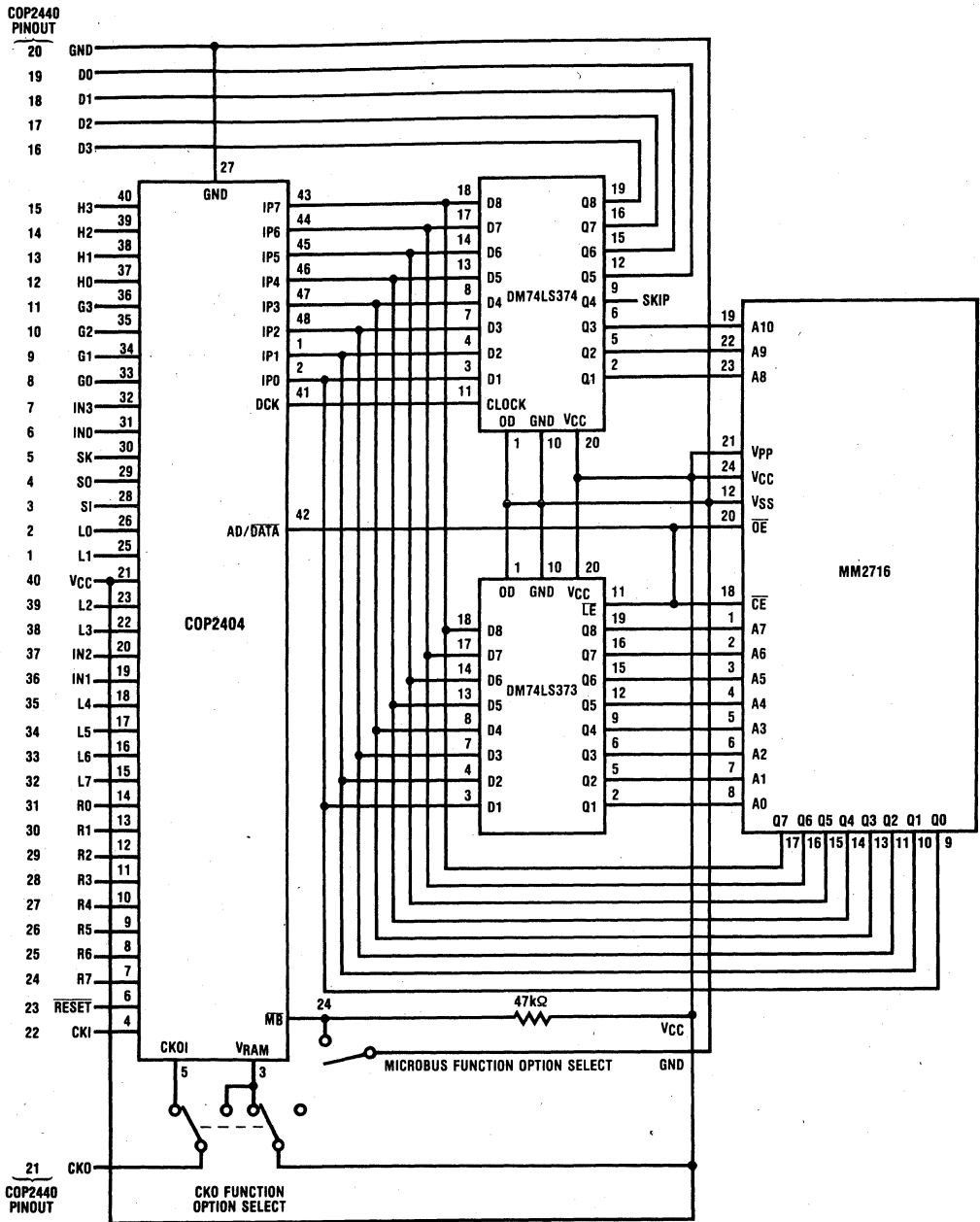


Figure 4. COP2404 Used to Emulate a COP2440

## Functional Description

The COP2404 is a ROMless microcontroller for emulating the COP2440 or for stand-alone applications. Please refer to the COP2440 description for detail functional description. The following describes functions that are unique to the COP2404 or are different from those in COP2440. All references to COP2404 also apply to COP2304. Figures 1 and 2 show the COP2404 block diagram and pin-out.

### Program Memory

Program memory consists of 2048 bytes of external memory (on-chip in the COP2440) that can be accessed through the IP port. See External Memory Interface below.

### D Port

The D3-D0 outputs are missing from this 48-pin package, but may be recovered through the IP port (see External Memory Interface below). Note that the recovered signals have the same timing but different output drive capability as those from the COP2440 (see D Port Characteristics below).

### MICROBUS™ and Zero-Crossing Detect Input Option

The MICROBUS compatible I/O, selected by a mask option on the COP2440, is selected by tying the MB pin directly to ground. When the MICROBUS compatible I/O is not desired, the MB pin should be tied to V<sub>CC</sub>. Note that none of the IN inputs are Hi-Z. Since zero-crossing detect input (used by INIL instruction and zero-crossing interrupt feature) is chosen for IN1, the IN1 input "1" level for ININ instruction, IN1 interrupt, and MICROBUS input is 3V. Even though the MICROBUS option and zero-crossing detector option appear on the COP2404, they are mutually exclusive on the COP2440.

### Oscillator

CKI is an external clock input signal. The clock frequency is divided by 16 to give the execution frequency.

### CKO Pin Options

Two different CKO functions of the COP2440 are available on the COP2404. V<sub>RAM</sub> supplies power to the lower 4 registers of RAM, and CKO1 is an interrupt input or a general purpose input, reading into bit 2 of A (accumulator) through the INIL instruction.

### External Memory Interface

The COP2404 is designed for use with an external program memory. This memory may be implemented using any devices having the following characteristics:

1. Random addressing
2. TTL-compatible TRI-STATE® outputs
3. TTL-compatible inputs
4. Access time = 450ns maximum

Typically these requirements are met using bipolar or MOS PROMS.

Suppose we are looking at the IP port when processor X is executing. While DCK is low, the upper three address bits, P10-P8, of the next instruction to be executed by processor X are sent out to IP2-IP0 respectively. D3-D0 are sent out to IP7-IP4. IP3 contains the SKIP output which is used by the COPS™ Program Development System (PDS). These data are clocked into D flip-flops by the rising edge of DCK. The timing of D port data is then the same for COP2404 and COP2440. After DCK goes to a "1" level, the remaining address bits (P7-P0) are sent out to IP7-IP0. They are latched into flow-through latches, e.g., 74LS373 when AD/DAT<sub>A</sub> goes low. The latched addresses provide the inputs to the external memory. When AD/DAT<sub>A</sub> goes low, the IP lines become program memory inputs from the external memory. Note that DCK has twice the cycle frequency of COP2404 with a duty cycle of about 50% and AD/DAT<sub>A</sub> has twice the cycle frequency with a duty cycle of about 75%. Figure 3 shows the timings for IP port, DCK and AD/DAT<sub>A</sub>. Figure 4 shows how to emulate the COP2440 using a COP2404 and an EPROM as the external memory.

### ROM Interface Timing Example

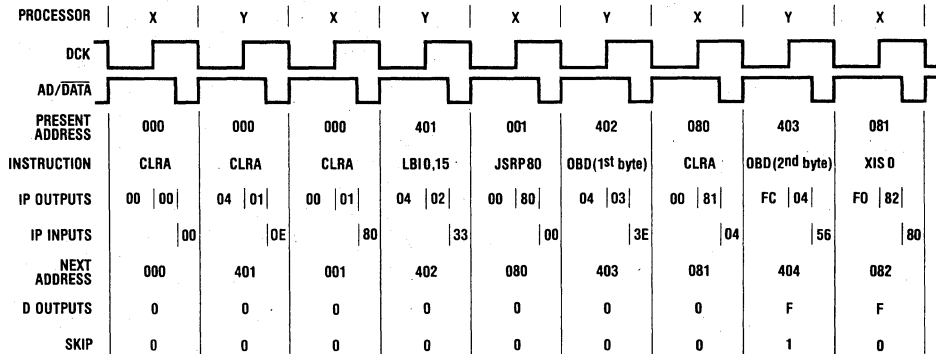
The following example shows the timing relationship between IP port I/O data (to and from external ROM) and the present instruction that is being executed by a processor. A sample program starts with the following instructions:

	OP		
	ADD CODE		
000	00	CLRA	;CLRA IS 1ST INSTRUCTION
	:		
	:		PROCESSOR X STARTS HERE
	:		
001	80	JSRP CLREG	;CLEAR REGISTER
	:		
	:		
	:		SUBROUTINE PAGE
	:		
080	00	CLREG: CLRA	
081	04	XIS	
082	80	JP -2	
083	48	RET	
	:		
	:		PROCESSOR Y STARTS HERE
	:		
401	0E	LBI 0,15	;OUTPUT 15 TO D
402	333E	OBD	
404	56	AISC 6	;PUT 6 TO ACCUMULATOR
	:		
	:		
	:		

Figure 5 shows what IP inputs and outputs are in relationship with the instructions that are being executed during the first few cycles of the above program.



## Timing Diagram (Continued)



NOTE: THE LAST 3 ROWS—NEXT ADDRESS, D OUTPUTS AND SKIP MAY BE DECODED FROM IP OUTPUTS

Figure 5. IP Port I/O Timing

### I/O Options

All inputs except IN1 and CKI have on-chip depletion load device to  $V_{CC}$ . IN1 has a resistive load to GND due to the zero-crossing input. CKI is a Hi-Z input.

G and H ports have standard outputs. L and R ports have TRI-STATE® outputs. IP port, DCK, AD/DATA, SO and SK have push-pull outputs.

### LED Drive

The TRI-STATE outputs of L port may be used to drive the segments of an LED display. External current limiting resistors of 100 ohms must be connected between the L outputs and the LED segments.

### D Port Characteristics

Since the D port is recovered through an external latch, the output drive is that of the latch and not that of COP2440. Using the set-up as shown in Figure 4, at an output "0" level of 0.4V, the 74LS374 may sink 10 times as much current as the COP2440. At an output "1" level of 2.4V, the 74LS374 may source 10 times as much current as the COP2440. On the other hand, the output "1" level of 74LS374 latch does not go to  $V_{CC}$  without an external pull-up resistor. In order to better approximate the COP2440 output characteristics, add a 74C906 buffer to the output of the 74LS374, thus emulating an open drain D output. A pull-up resistor of 10k should be added to the input of the buffer. To emulate the standard output, add a pull-up resistor between 2.7k and 15k to the output of the 74C906.

### COP2404 Mask Options

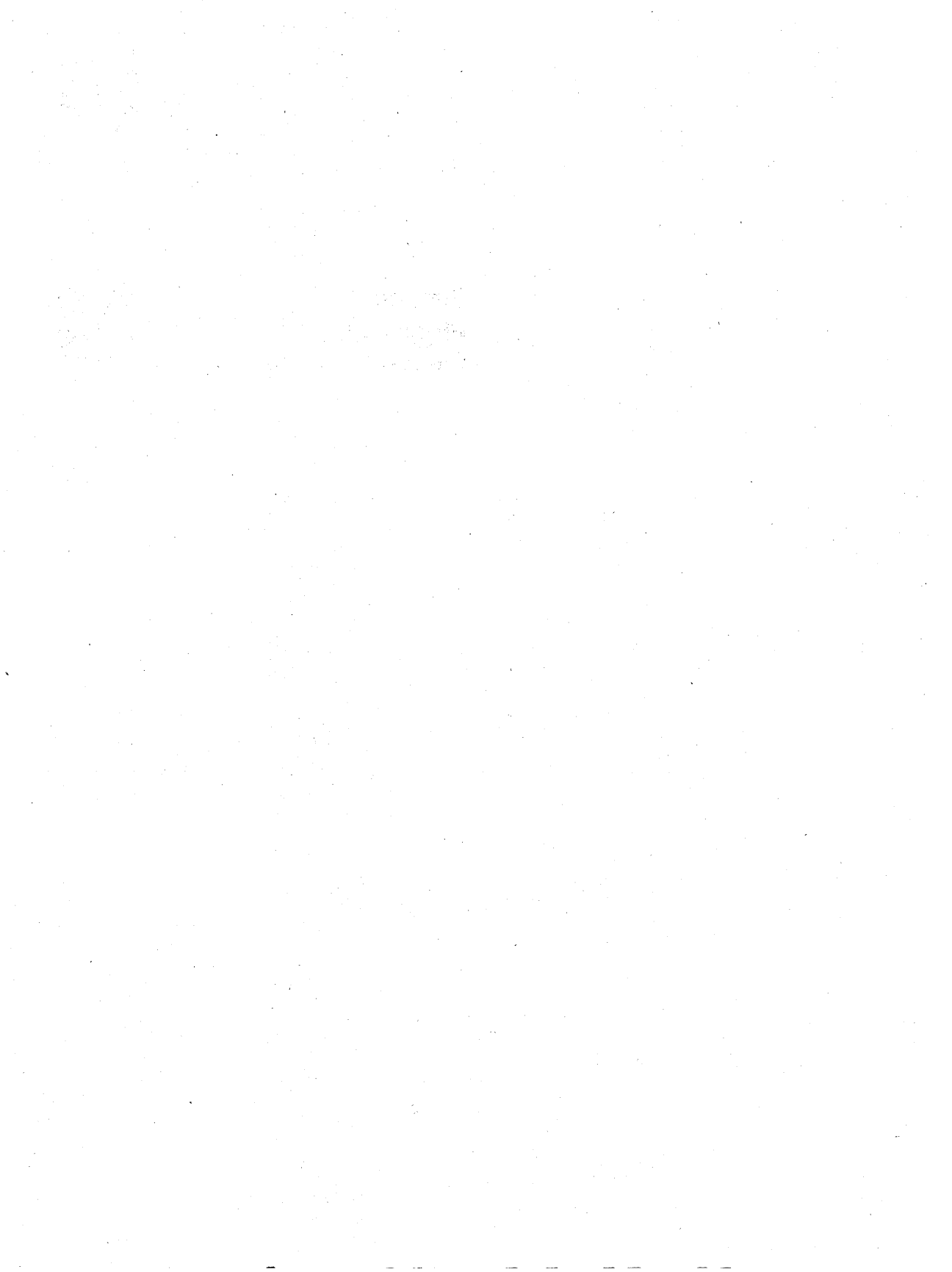
The following COP2440 options have been implemented in the COP2404.

Option Value	Comment
Option 1-2 = 3	L outputs are TRI-STATE®
Option 3 = 0	SI has load to $V_{CC}$
Option 4 = 2	SO is push-pull output
Option 5 = 2	SK is push-pull output
Option 6 = 0	IN0 has load to $V_{CC}$
Option 7 = 0	IN3 has load to $V_{CC}$
Option 8-11 = 0	G outputs are standard
Option 12-15 = 0	H outputs are standard
Option 16-19 = N/A	D outputs are derived from external latch, see Figure 4
Option 20 = N/A	GND — No option
Option 21 = 1,2	CKO is replaced by $V_{RAM}$ and CKOI
Option 22 = 0	CKI is input clock divided by 16
Option 23 = 0	RESET has load to $V_{CC}$
Option 24-31 = 3	R outputs are TRI-STATE
Option 32-35 = 3	L outputs are TRI-STATE
Option 36 = 2	IN1 is zero-crossing detect input
Option 37 = 0	IN2 has load to $V_{CC}$
Option 38-39 = 3	L outputs are TRI-STATE
Option 40 = N/A	$V_{CC}$ — No option available
Option 41 = 0,1	MICROBUS option is pin selectable
Option 42-48 = 0	Inputs have standard TTL levels
Option 49 = N/A	No option available
Option 50 = N/A	48-pin package



**Section 4**  
**Piggyback**  
**Microcontrollers**

**4**



# COP420R/COP444LR Piggyback-EPROM Microcontroller

## General Description

The COP420R and COP444LR Piggyback-EPROM microcontrollers are members of the COP<sup>SM</sup> family. The COP420R and COP444LR devices are identical to the COP420 and COP444L respectively except that the program ROM has been removed. In place of the ROM each device package incorporates the circuitry and socket to accommodate the Piggyback-EPROM.

The socket provided on the package accepts an MM2716, NMC27C16, MM2758A, or MM2758B EPROM. Each part is a complete microcontroller system with CPU, RAM, I/O, and EPROM socket provided in a single 28-pin package. In a system the COP420R and COP444LR will perform exactly as its mask programmed equivalent.

The complete package allows field test of a system in its final electrical and mechanical configuration. This important benefit facilitates development and debug of a COP400 program prior to masking of a production part.

These devices are also economical in low and medium volume applications or when the program may require changing.

COPS and MICROWIRE are trademarks of National Semiconductor Corp.

## Features

- Exact equivalent of the KOP420 and COP444L — plugs into same socket
- Socket and interface for industry standard EPROMs
- Self-contained voltage regulator for EPROM on COP444LR
- Powerful instruction set
- True vectored interrupt, plus restart
- Three-level subroutine stack
- Compatible with all COPS family peripherals
- Internal binary counter register with MICROWIRE<sup>SM</sup> family peripherals compatible serial I/O
- Software and hardware compatible with other members of the COPS family
- Single supply operation
- Internal presettable time base counter for real time processing
- 4  $\mu$ s instruction time (COP420R)
- 16  $\mu$ s instruction time (COP444LR)
- 23 I/O lines

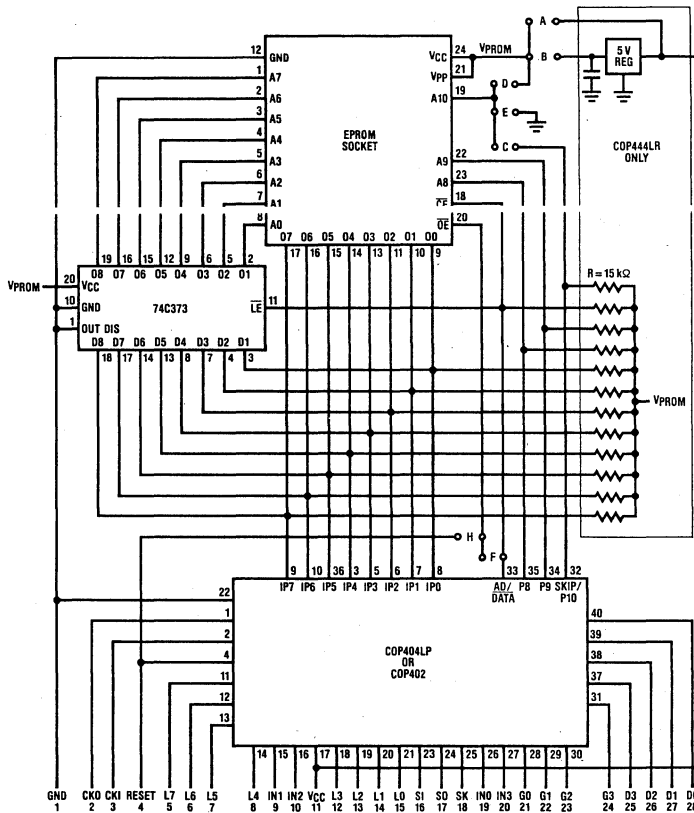


Figure 1. COP420R/COP444LR Block Diagram

COP420R/COP444LR



## COP420R

### Absolute Maximum Ratings

Voltage at any Pin	-0.3V to +7.0V
Operating Temperature Range	0°C to 70°C
Storage Temperature Range	-65°C to 150°C
Lead Temperature (soldering, 10 sec.)	300°C
Package Power Dissipation	see Figure 15
Total Sink Current	50 mA
Total Source Current	70 mA

*Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

### DC Electrical Characteristics 0°C to 70°C, 4.5V to 6.3V unless otherwise noted

Parameter	Conditions	Min.	Max.	Units
Operation Voltage		4.5	6.3	V
Power Supply Ripple	(peak to peak) note 3		0.4	V
Supply Current	All outputs open, no EPROM installed		38	mA
Input Voltage Levels				
CKI Input Levels				
Crystal Input				
Logic High		2.0		V
Logic Low		-0.3	0.4	V
Schmitt Trigger Input				
RESET				
Logic High		0.7V <sub>CC</sub>		V
Logic Low		-0.3	0.6	V
All Other Inputs				
Logic High	V <sub>CC</sub> = Max.	3.0		V
Logic High	V <sub>CC</sub> = 5.0V ± 5%	2.0		V
Logic Low		-0.3	0.8	V
Input Load Source Current	V <sub>CC</sub> = 5.0V, V <sub>IN</sub> = 0	-100	-800	μA
Input Capacitance			7.0	pF
Hi-Z Input Leakage	V <sub>CC</sub> = 5.0V	-1.0	+1.0	μA
Output Voltage Levels				
D, G, L, SK, SO Outputs				
TTL Operation	V <sub>CC</sub> = 5.0V ± 5%			
Logic High	I <sub>OH</sub> = -100 μA	2.4		V
Logic Low	I <sub>OL</sub> = 1.6 mA	-0.3	0.4	V
A <sub>9</sub> -A <sub>0</sub> , CKO Outputs				
Logic High	I <sub>OH</sub> = -75 μA	2.4		V
Logic Low	I <sub>OL</sub> = 400 μA	-0.3	0.4	V
CMOS Operation				
Logic High	I <sub>OH</sub> = -10 μA	V <sub>CC</sub> - 1		V
Logic Low	I <sub>OL</sub> = 10 μA	-0.3	0.2	V
Output Current Levels				
LED Direct Drive (COP402)				
Logic High	V <sub>CC</sub> = 6.0V V <sub>OH</sub> = 2.0V	2.5	14	mA
Allowable Sink Current				
Per Pin (L, D, G)			10	mA
Per Pin (all others)			2.0	mA
Per Port (L)			16	mA
Per Port (D, G)			10	mA
Allowable Source Current				
Per Pin (L)			-15	mA
Per Pin (all others)			-1.5	mA

## AC Electrical Characteristics 0°C to 70°C, 4.5V to 6.3V unless otherwise noted

Parameter	Conditions	Min.	Max.	Units
Instruction Cycle Time		4.0	10	μs
Operating CKI Frequency	÷16 mode	1.6	4.0	MHz
CKI Duty Cycle (note 1)		40	60	%
Rise Time	Frequency = 4.0 MHz		60	ns
Fall Time	Frequency = 4.0 MHz		40	ns
<b>Inputs:</b>				
SI				
t <sub>SETUP</sub>		0.3		μs
t <sub>HOLD</sub>		250		ns
All other Inputs				
t <sub>SETUP</sub>		1.7		μs
t <sub>HOLD</sub>		300		ns
Output Propagation Delay				
SO and SK				
t <sub>PD1</sub>	Test Conditions: R <sub>L</sub> = 5.0k, C <sub>L</sub> = 50 pF, V <sub>OUT</sub> = 1.5V		1.0	μs
t <sub>PD0</sub>			1.0	μs
CKO				
t <sub>PD1</sub>			0.25	μs
t <sub>PD0</sub>			0.25	μs
AD/ <u>DATA</u>				
t <sub>PD1</sub>			0.6	μs
t <sub>PD0</sub>			0.6	μs
A <sub>7</sub> -A <sub>0</sub>				
t <sub>PD1</sub>			2.0	μs
t <sub>PD0</sub>			2.0	μs
All other Outputs				
t <sub>PD1</sub>			1.5	μs
t <sub>PD0</sub>			1.5	μs

**Note 1:** Duty cycle =  $tW1/(tW1 + tW0)$ .

**Note 2:** See Figure 6 for additional I/O characteristics.

**Note 3:** Voltage change must be less than 0.5 volts in a 1 ms period.

**Note 4:** Exercise great care not to exceed maximum device power dissipation limits when direct driving LEDs (or sourcing similar loads) at high temperature.

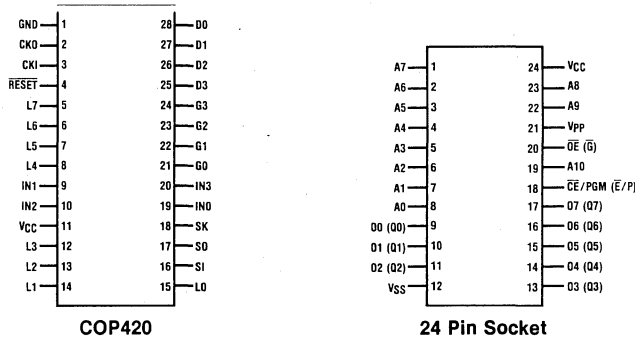


Figure 2. COP420R Connection Diagrams

Pin	Description	Pin	Description
L <sub>7</sub> -L <sub>0</sub>	8 bidirectional I/O ports with TRI-STATE®	AD/ <u>DATA</u>	Address out/data in flag
G <sub>3</sub> -G <sub>0</sub>	4 bidirectional I/O ports	CKI	System oscillator input
D <sub>3</sub> -D <sub>0</sub>	4 general purpose outputs	CKO	General purpose input
IN <sub>3</sub> -IN <sub>0</sub>	4 general purpose inputs	RESET	System reset input
SI	Serial input (or counter input)	VCC	Power supply
SO	Serial output (or general purpose output)	GND	Ground
SK	Logic-controlled clock (or general purpose output)	O <sub>7</sub> -O <sub>0</sub>	PROM data lines
		A <sub>9</sub> -A <sub>0</sub>	PROM address outputs



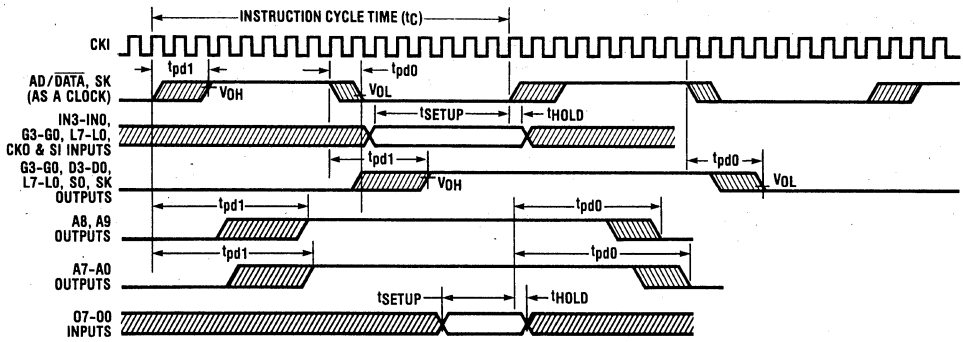


Figure 3a. COP420R Input/Output Timing Diagrams (Crystal ÷ 16 Mode)

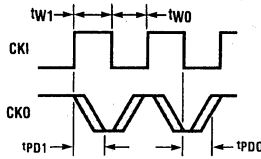
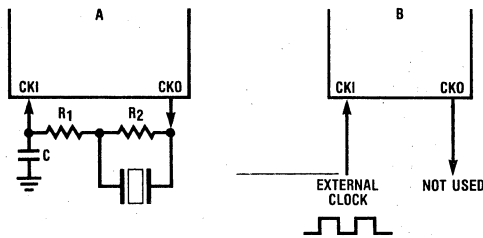


Figure 3b. COP420R CKO Output Timing

### Oscillator

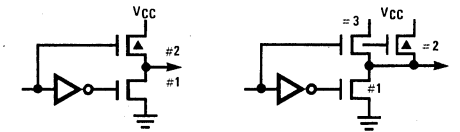
There are two basic clock oscillator configurations available for the COP420R as shown by Figure 4.

- a. **Crystal Controlled Oscillator.** CKI and CKO are connected to an external crystal. The instruction cycle time equals the crystal frequency divided by 16.
- b. **External Oscillator.** CKI is driven by an external clock signal. The instruction cycle time is the clock frequency divided by 16.



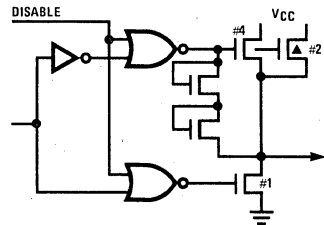
Crystal Value	Component Values		
	R1	R2	C
4 MHz	1k	1M	27pF
3.58 MHz	1k	1M	27pF
2.09 MHz	1k	1M	56pF

Figure 4. COP420R Oscillator

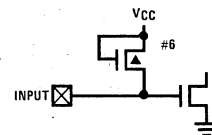


D<sub>3</sub>-D<sub>0</sub>, G<sub>3</sub>-G<sub>0</sub> Standard Output

S<sub>0</sub>, S<sub>K</sub> Push-Pull Output



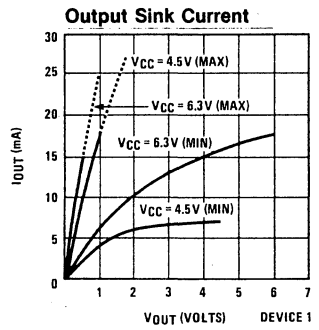
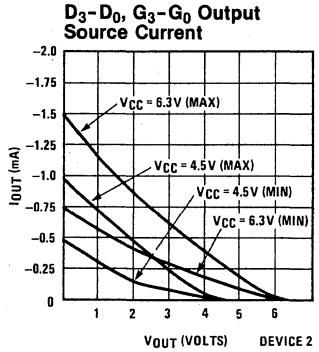
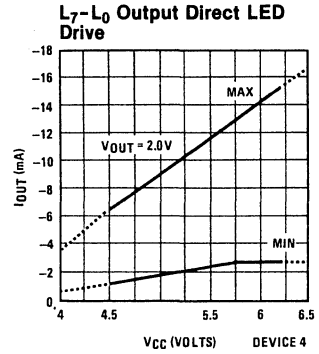
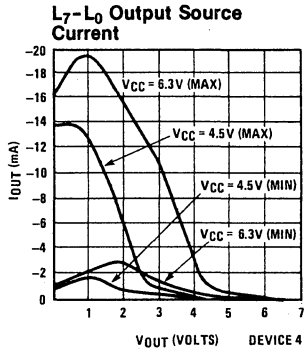
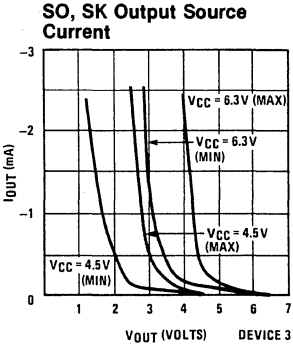
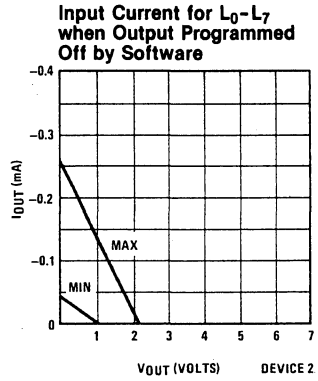
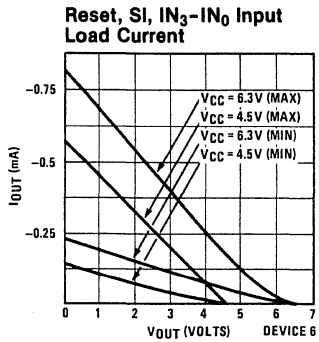
L<sub>7</sub>-L<sub>0</sub> LED Output



Reset, SI, IN<sub>3</sub>-IN<sub>0</sub> Input with Load

(▲ IS DEPLETION DEVICE)

Figure 5. COP420R Input/Output Configurations



Note: Absolute maximum ratings for the COP420R must be observed

Figure 6. COP420R Input/Output Characteristics

**COP444LR****Absolute Maximum Ratings**

Voltage at Any Pin Relative to GND	-0.5V to +10V
Ambient Operating Temperature	0°C to 70°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C
Power Dissipation	see Figure 15
Total Source Current	120 mA
Total Sink Current	140 mA

*Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

**DC Electrical Characteristics**  $0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 9.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Operating Voltage ( $V_{CC}$ )	(Notes 1 and 2)	4.5	9.5	V
Power Supply Ripple	Peak to Peak		0.5	V
Operating Supply Current	(All inputs and outputs open) No EPROM Installed		25	mA
Input Voltage Levels				
CKI Input Levels				
Crystal Input				
Logic High ( $V_{IH}$ )		2.0		V
Logic Low ( $V_{IL}$ )		-0.3	0.4	V
RESET Input Levels	(Schmitt Trigger Input)			
Logic High		$0.7V_{CC}$		V
Logic Low		-0.3	0.6	V
$O_0$ - $O_7$ , SI Input Levels				
Logic High	$V_{CC} = 9.5\text{V}$	2.4		V
Logic High	$V_{CC} = 5.0\text{V} \pm 5.0\%$	2.0		V
Logic Low		-0.3	0.8	V
All Other Inputs				
Logic High	High Trip Level Options Selected	3.6		V
Logic Low		-0.3	1.2	V
Input Capacitance			7.0	pf
Output Voltage Levels				
LSTTL Operation	$V_{CC} = 5.0\text{V} \pm 5.0\%$			
Logic High ( $V_{OH}$ )	$I_{OH} = -25\mu\text{A}$	2.7		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 0.36\text{mA}$		0.4	V
Output Current Levels (Note 3)				
Output Sink Current ( $I_{OL}$ )				
SO and SK Outputs	$V_{CC} = 9.5\text{V}$ , $V_{OL} = 0.4\text{V}$	1.8		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.9		mA
$L_0$ - $L_7$ Outputs	$V_{CC} = 9.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.8		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.4		mA
$G_0$ - $G_3$ and $D_0$ - $D_3$ Outputs	$V_{CC} = 9.5\text{V}$ , $V_{OL} = 1.0\text{V}$	30		mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 1.0\text{V}$	15		mA

**COP444LR****DC Electrical Characteristics** (Cont'd.)  $0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 9.5\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Output Source Current ( $I_{OH}$ )				
D <sub>0</sub> -D <sub>3</sub> , G <sub>0</sub> -G <sub>3</sub> Outputs	$V_{CC} = 9.5\text{V}$ , $V_{OH} = 2.0\text{V}$ $V_{CC} = 4.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-140 -30	-800 -250	$\mu\text{A}$ $\mu\text{A}$
SO and SK Outputs	$V_{CC} = 9.5\text{V}$ , $V_{OH} = 4.75\text{V}$ $V_{CC} = 4.5\text{V}$ , $V_{OH} = 1.0\text{V}$	-1.4 -1.2		$\text{mA}$ $\text{mA}$
L <sub>0</sub> -L <sub>7</sub> Outputs	$V_{CC} = 9.5\text{V}$ , $V_{OH} = 2.0\text{V}$ $V_{CC} = 6.0\text{V}$ , $V_{OH} = 2.0\text{V}$	-3.0 -3.0	-30 -20	$\text{mA}$ $\text{mA}$
Input Load Source Current ( $I_{IL}$ )	$V_{CC} = 5.0\text{V}$ , $V_{OH} = 0\text{V}$	-10	-140	$\mu\text{A}$
Total Sink Current Allowed				
All Outputs Combined			140	$\text{mA}$
D, G Ports			120	$\text{mA}$
L <sub>7</sub> -L <sub>4</sub>			4.0	$\text{mA}$
L <sub>3</sub> -L <sub>0</sub>			4.0	$\text{mA}$
All Other Pins			1.8	$\text{mA}$
Total Source Current Allowed				
All I/O Combined			120	$\text{mA}$
L <sub>7</sub> -L <sub>4</sub>			60	$\text{mA}$
L <sub>3</sub> -L <sub>0</sub>			60	$\text{mA}$
Each L Pin			30	$\text{mA}$
All Other Pins			1.5	$\text{mA}$

**COP444LR****AC Electrical Characteristics**  $0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 9.5\text{V}$  unless otherwise specified.

Parameter	Conditions	Min.	Max.	Units
Instruction Cycle Time		15	40	$\mu\text{s}$
CKI				
Input Frequency $f_i$	( $\div 32$ mode)	0.8	2.1	$\text{MHz}$
Duty Cycle		30	60	%
Rise Time	$f_i = 2.097\text{MHz}$		120	$\text{ns}$
Fall Time			80	$\text{ns}$
Inputs				
SI, IP7-IP0				
$t_{\text{SETUP}}$		2.0		$\mu\text{s}$
$t_{\text{HOLD}}$		1.0		$\mu\text{s}$
IN <sub>3</sub> -IN <sub>0</sub> , G <sub>3</sub> -G <sub>0</sub> , L <sub>7</sub> -L <sub>0</sub>				
$t_{\text{SETUP}}$		8.0		$\mu\text{s}$
$t_{\text{HOLD}}$		1.3		$\mu\text{s}$
Output Propagation Delay	Test Condition: $C_L = 50\text{pF}$ , $V_{\text{OUT}} = 1.5\text{V}$ , $R_L = 20\text{k}\Omega$			
SO, SK Outputs			4.0	$\mu\text{s}$
$t_{\text{PD1}}$ , $t_{\text{PD0}}$				
D <sub>3</sub> -D <sub>0</sub> , G <sub>3</sub> -G <sub>0</sub> , L <sub>7</sub> -L <sub>0</sub>			5.6	$\mu\text{s}$
$t_{\text{PD1}}$ , $t_{\text{PD0}}$				
A <sub>0</sub> -A <sub>7</sub>			7.5	$\mu\text{s}$
$t_{\text{PD1}}$ , $t_{\text{PD0}}$				
A <sub>8</sub> , A <sub>9</sub>			11.5	$\mu\text{s}$
$t_{\text{PD1}}$ , $t_{\text{PD0}}$				
A <sub>10</sub>			6.0	$\mu\text{s}$
$t_{\text{PD1}}$ , $t_{\text{PD0}}$				

Note 1: See section on  $V_{CC}$  considerations.

Note 2:  $V_{CC}$  voltage changes must be less than 0.5V/ms to maintain proper operation.

Note 3: See Figure 11 for additional I/O characteristics.

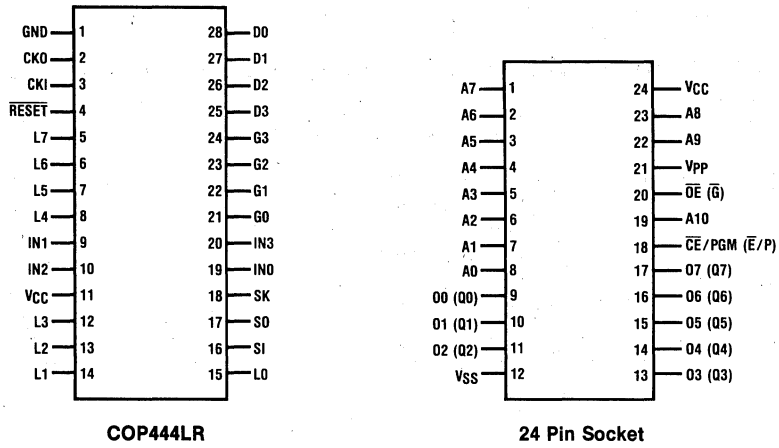


Figure 7. COP444LR Connection Diagram

Pin	Description	Pin	Description
L7-L0	8 bidirectional I/O ports with TRI-STATE®	AD/DATA	Address out/data in flag
G3-G0	4 bidirectional I/O ports	CKI	System oscillator input
D3-D0	4 general purpose outputs	CKO	General purpose input
IN3-IN0	4 general purpose inputs	RESET	System reset input
SI	Serial input (or counter input)	VCC	Power supply
SO	Serial output (or general purpose output)	GND	Ground
SK	Logic-controlled clock (or general purpose output)	O7-O0	PROM data lines
		A10-A0	PROM address outputs

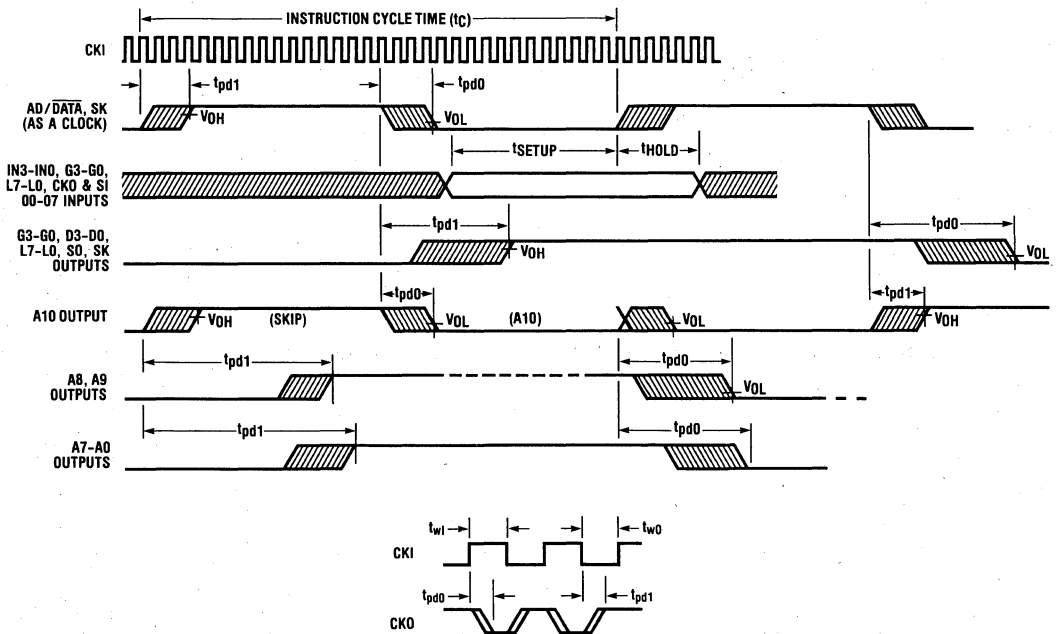


Figure 8. COP444LR Input/Output Timing Diagram

**Oscillator**

CKI is an external clock input signal. The external frequency is divided by 32 to give the instruction cycle time.

**CKO as General Purpose Input**

CKO has been configured as a general purpose input with a load device to  $V_{CC}$ . The logic level applied to CKO will be read into bit 2 of A (accumulator) upon execution of an INIL instruction.

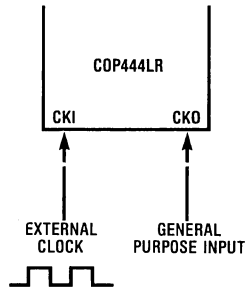
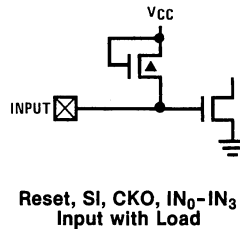
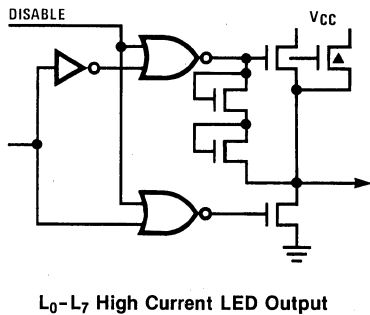
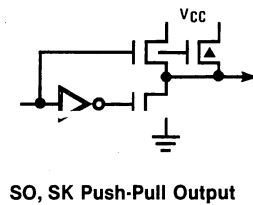
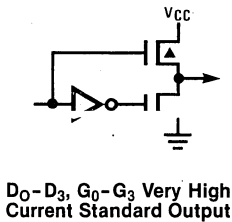


Figure 9. COP444LR Oscillator

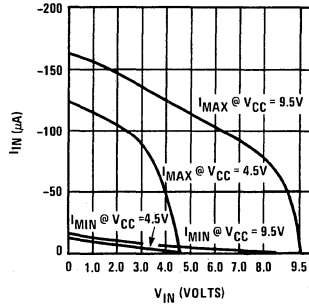


(▲ IS DEPLETION DEVICE)

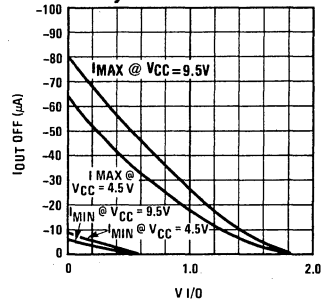
Figure 10. COP444LR Input/Output Configurations



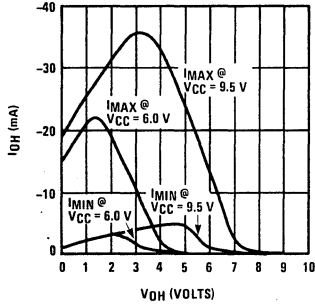
Reset, SI, CKO, IN<sub>0</sub>-IN<sub>3</sub>  
Input Load Current



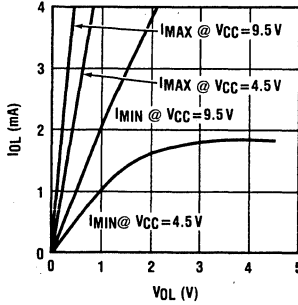
Input Current for L<sub>0</sub>-L<sub>7</sub>  
when Output Programmed  
Off by Software



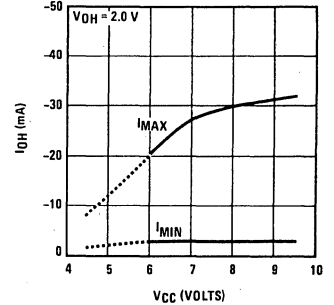
L<sub>0</sub>-L<sub>7</sub> Output Source  
Current



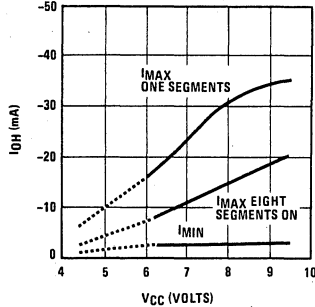
L<sub>0</sub>-L<sub>7</sub> Output Sink  
Current



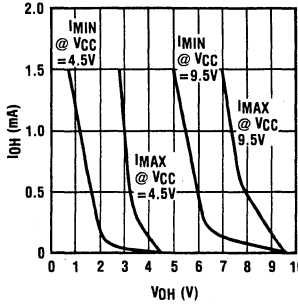
L<sub>0</sub>-L<sub>7</sub> Output Direct LED  
Drive



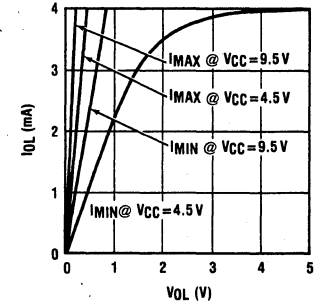
L<sub>0</sub>-L<sub>7</sub> Direct Segment Drive  
and D<sub>0</sub>-D<sub>3</sub> or G<sub>0</sub>-G<sub>3</sub>  
Direct Digit Drive



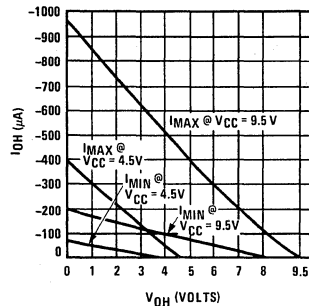
SO, SK Output Source  
Current



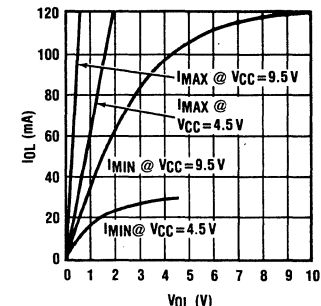
SO, SK Output Sink  
Current



D<sub>0</sub>-D<sub>7</sub>, G<sub>0</sub>-G<sub>3</sub> Output  
Source Current



D<sub>0</sub>-D<sub>3</sub>, G<sub>0</sub>-G<sub>3</sub> Output Sink  
Current



Note: Absolute maximum ratings for the COP444LR must be observed.

Figure 11. COP444LR Input/Output Characteristics

**External Memory Interface**

The COP420R/COP444LR are designed for use with an external program memory. This memory may be implemented using the EPROMs listed in Table 1.

**Table 1. EPROMs for use with COP420R/COP444LR**

V <sub>CC</sub> of EPROM	COP420R	COP444LR
5.0V ± 0.25V	MM2716 NMC27C16 MM2758A MM2758B	MM2716 NMC27C16
5.0V ± 0.5V	MM2716-1 MM2716E MM2716M NMC27C16-1	MM2716-1 MM2716E MM2716M NMC27C16-1

**Table 2. Jumper Configurations**

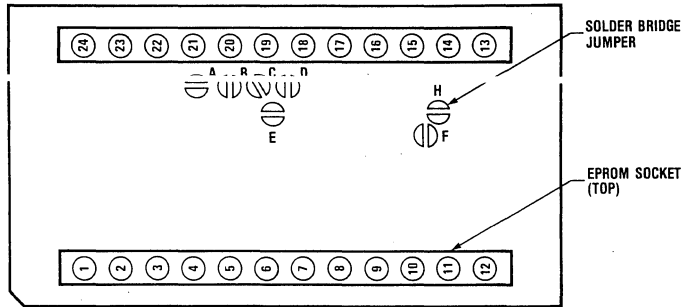
Jumper	Function
A	Circuit V <sub>CC</sub> connected directly to EPROM's V <sub>CC</sub> pin (V <sub>prom</sub> )
B	Circuit V <sub>CC</sub> connected to regulator input, regulator output connected to EPROM's V <sub>CC</sub> pin (V <sub>prom</sub> )
C	EPROM A10 pin connected to COP404L
D	EPROM A10 pin connected to V <sub>CC</sub> (MM2758A)
E	EPROM A10 pin connected to GND (MM2758B)
F	EPROM $\overline{OE}$ pin connected to AD/ $\overline{DATA}$
H	Not used

**Jumper Configurations**

In order to enable various options seven solder bridge type jumpers, labeled A through H, have been implemented. These jumpers are located in the area underneath the EPROM. See Figure 12 and Table 2.

The COP420R is shipped with jumpers A, E, and F installed.

The COP444LR is shipped with jumpers B, C, and F installed.



**Figure 12. Jumper Locations**

**General V<sub>CC</sub> Considerations**

The CPU portion of the COP420R is the COP402. The V<sub>CC</sub> operating range for the COP402 is 4.5V to 6.3V. The CPU portion of the COP444LR is the COP404LP. The V<sub>CC</sub> operating range for the COP444LP is 4.5V to 9.5V.

Due to the fact that the V<sub>CC</sub> operating range for the EPROMs is either 4.75V to 5.25V or 4.5V to 5.5V the EPROMs become the V<sub>CC</sub> limiting device. Because of these limitations jumpers have been added on the COP420R; jumpers and a regulator have been added on the COP444LR.

A 0.1μF decoupling capacitor should be connected between V<sub>CC</sub> and Ground as close to the device as possible.

**V<sub>CC</sub> Considerations for the COP420R**

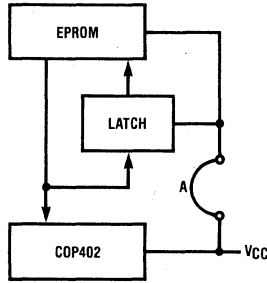
In the COP420R, jumper A is connected (Figure 13a). With A in this configuration the V<sub>CC</sub> operating range becomes the V<sub>CC</sub> operating range of the EPROM selected.

If the jumper at A is replaced by a diode the V<sub>CC</sub> operating range will be changed. For example, if the diode voltage is 0.8V and the EPROM selected is 4.5V to 5.5V the operating range of the COP420R becomes:

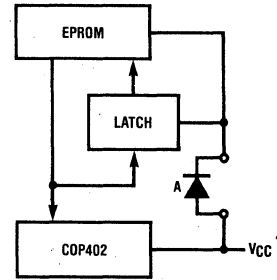
$$4.5V + 0.8V \text{ to } 5.5V + 0.8V \text{ or } 5.3V \text{ to } 6.3V.$$

**WARNING: THIS CHANGE SHOULD BE MADE WITH EXTREME CAUTION. IMPROPER INSTALLATION VOIDS WARRANTY.**

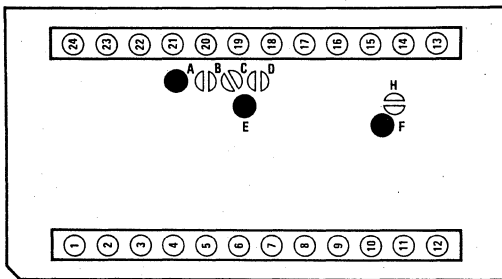
Remove solder from jumper A and insert the anode of the diode through the hole connected to the bottom of the jumper A and the cathode of the diode through the hole connected to the top of jumper A as shown in Figure 13B.



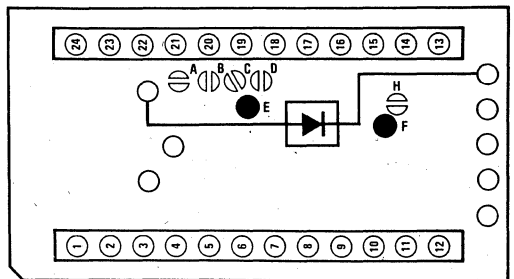
a. Standard Configuration



b. Diode Configuration



c. Jumpers for Standard Configuration as Shipped from NSC



d. Jumpers for Diode Configuration as Modified by User

Figure 13. COP420R Jumper Connections

**V<sub>CC</sub> Considerations for the COP444LR**

In the COP444LR, jumper B is connected (Figure 14). With B in this configuration the 5.0V regulator is connected to the EPROM and the latch. The V<sub>CC</sub> range of the COP444LR is then determined by the V<sub>IN</sub>/V<sub>OUT</sub> specification of the regulator; which is 2.0V. Therefore, the V<sub>CC</sub> range of the COP444LR is 7.0v to 9.5V.

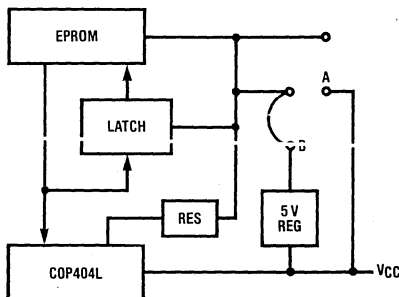
If the jumper is removed from the B position and connected in the A position, the same limitations apply that are discussed above in the section on V<sub>CC</sub> Considerations for the COP420R.

**WARNING: THIS CHANGE SHOULD BE MADE WITH EXTREME CAUTION. IMPROPER INSTALLATION VOIDS WARRANTY.**

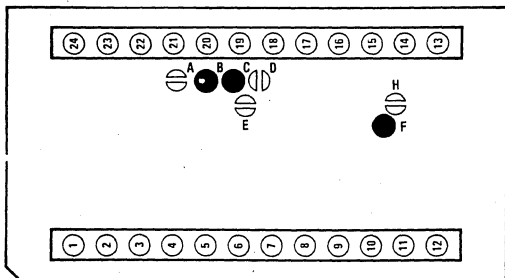
**Power Considerations for COP420R/COP444LR**

The absolute maximum power dissipation of the COP420R and the COP444LR is shown in Figure 15. In addition, the COP444LR contains a regulator with an absolute maximum power dissipation of 305 mW at 70°C.

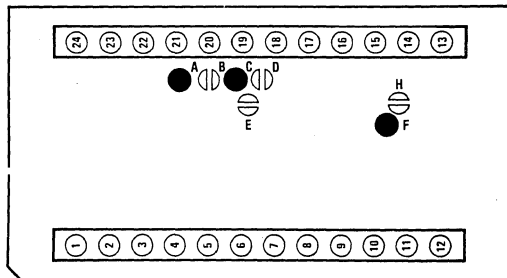
For an MM2716 EPROM the maximum operating current is 105mA and the maximum current in the standby mode is 30mA. The COP444LR is designed such that the EPROM is in the standby mode for 50% of the time. Therefore the power consumed by the regulator is:  $(9.5 - 5.0)(105 + 30)/2 = 304 \text{ mW}$ .



a. Regulator Configuration



b. COP444LR Jumpers for V<sub>CC</sub> = 7.0 to 9.5V as Shipped from NSC



c. COP444LR Jumpers for V<sub>CC</sub> = 4.5 to 5.5V as Modified by User

Figure 14. COP444LR Jumper Connections

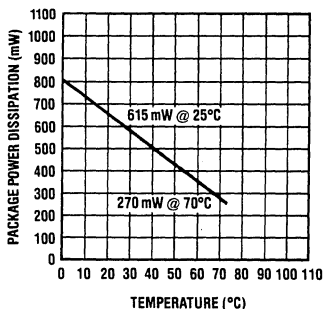


Figure 15. Maximum Power Dissipation for the COP420R/COP444LR

For the absolute maximum power dissipation of the COPS devices, all sources of power dissipation must be taken into account. For example:

When the COP outputs are used to drive loads directly the power consumed in the outputs must be considered in the maximum power dissipation of the package. Figure 16a shows an LED segment obtaining its source current from the L0 output and D0 sinking that current. In this configuration all the power required to drive the LED, with the exception of the portion consumed by the LED itself, is consumed within the chip. Assuming that the COP444LR is the driving device, Figure 11 shows the currents available on these outputs.

If we assume the  $V_{SOURCE}$  resistor is not inserted, the device has a  $V_{CC}$  of 9.5V, and that the voltage drop across the LED is 2.0V we can calculate the power dissipation in these outputs. The minimum current that D0 can sink at 1.0V is 35 mA. L0 can source up to 35 mA at 3.0V. Therefore, the power dissipation for the L0 output could be:  $(9.5 - 3.0)0.035 = 227$  mW. The power in the D0 output could be:  $1(0.035) = 35$  mW.

Figure 16b depicts the D0 output driving the base of a PNP transistor with a current limiting resistor. Without the current limiting resistor the absolute maximum sink current of the D0 output would be exceeded.

### Current Limiting Resistor Calculations

In order to calculate the current limiting resistor for the case shown in Figure 16a, LED Drive, we must refer to Figure 11, L0-L7 output source current. This figure shows that at  $V_{CC} = 9.5V$  the minimum current curve peaks at  $I = 6.0$  mA and  $V_{SOURCE} = 4.8V$ . The current curve is actually very flat between 4.0 and 5.0 volts. For maxi-

imum current we need to set the voltage on the L pin = 4.8V at 6.0 mA. The D line will sink this current at 0.4V. Therefore, the resistor and LED must make up the difference.

$$\begin{aligned} V_I &= V_D + IR + V_{LED} \\ 4.8 &= 0.4 + 0.006R + 2.0 \\ 2.4 &= 0.006R \\ R &= 400 \Omega \end{aligned}$$

At the other end of the curve, when the L line sources the maximum current, assume the LED and the D line will have the same voltage drop.

$$\begin{aligned} V_I &= 0.4 + IR + 2.0 \\ V_I &= 2.4 + IR \end{aligned}$$

From the curve of Figure 11 we see that at 6.4V the L line will source 10 mA. Therefore:  $V_I = 2.4 + 0.01(400) = 6.4V$ .

In the case of the D line driving the base of the PNP in Figure 16b, let us assume the 420R with a  $V_{CC}$  of 4.5V, a base to emitter resistor of 5.1 k $\Omega$ ,  $V_{BE} = 1.0V$ , and a worst case base drive requirement of 3.0 mA. We see that we must supply 200  $\mu A$  to the base-emitter resistor to turn the transistor on.

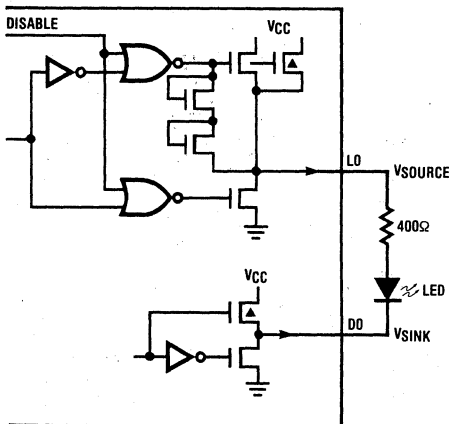
$$1.0V/5.1k\Omega = 200 \mu A$$

From Figure 6 we see that at 1.0V the D line can sink 3.2 mA. To calculate the value of the current limiting resistor we have:

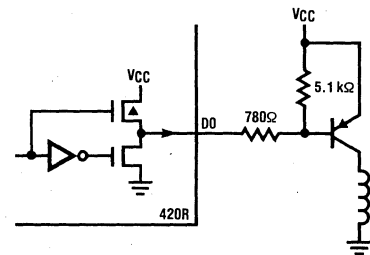
$$\begin{aligned} R &= (V_{CC} - V_{BE} - V_D)/I \\ R &= (4.5 - 1.0 - 1.0)/0.0032 = 780 \Omega \end{aligned}$$

At 6.3V the D line can sink more than enough current at 0.3V, and if the  $V_{BE}$  is 0.7V we can calculate the maximum D line current:

$$\begin{aligned} I &= (V_{CC} - V_{BE} - V_D)/R \\ I &= (6.3 - 0.7 - 0.3)/780 = 6.3 \text{ mA} \end{aligned}$$



a. LED Drive



b. PNP Drive

Figure 16. COP Output Loading

## Emulation of Other Members of the COPS™ Family

The pin configurations for members of the COPS family of microcontrollers are shown in Figure 17.

The COP420R, with an EPROM, is an exact emulator for the COP420. With appropriate pin scramblers, the COP420R will faithfully emulate the COP421 and COP422.

The COP444LR, with an EPROM, is an exact emulator for the COP444L. With a pin scrambler, the COP444LR will emulate the COP445L.

The COP444LR will emulate the COP420L if the limitations on ROM and RAM are observed. Also, with appropriate pin scramblers, the COP444LR will emulate the COP421L and COP422L.

The COP444LR can be used to emulate the COP410L and COP411L with a pin scrambler, but caution must be used. The COP410L and the COP411L not only have less ROM but the RAM registers are organized differently and the stack only has two (2) levels.

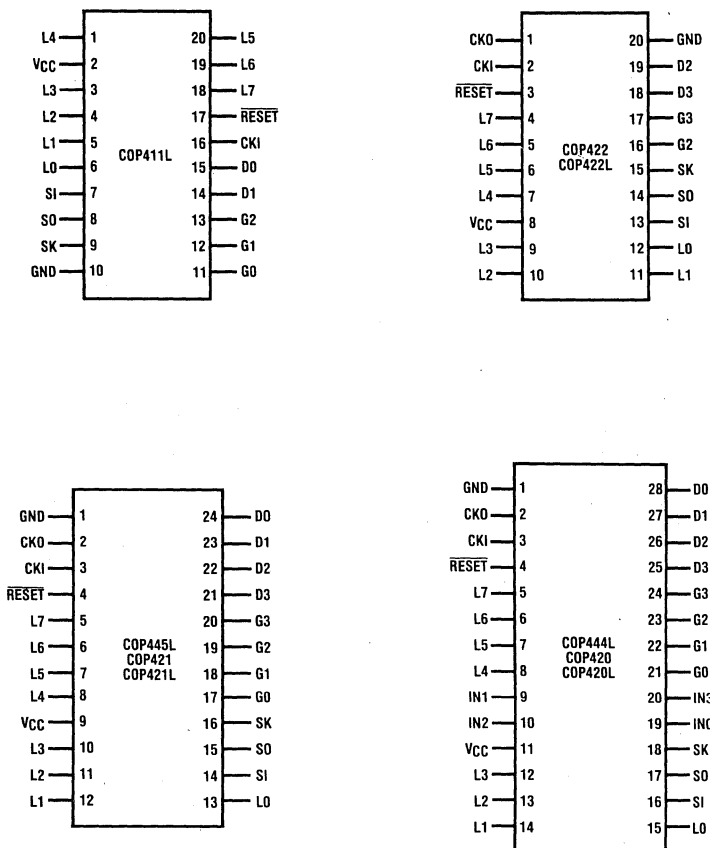


Figure 17. COPS Family Pin Configurations

# COP440R/COP2440R Piggyback-EPROM Microcontroller

## General Description

The COP440R/2440R Piggyback-EPROM Microcontrollers are members of the COPSTM family. The COP440R and COP2440R devices are identical to the COP440 and COP2440 respectively except that the program ROM has been removed. In place of the ROM, each device package incorporates the circuitry and socket to accommodate the Piggyback-EPROM.

The socket provided on the package accepts an MM2716 or NMC27C16. Each part is a complete microcontroller system with CPU, RAM, I/O, and EPROM socket provided in a single 40-pin package. In a system, the piggyback device will perform exactly as its mask-programmed equivalent.

The complete package allows field test of a system in its final electrical and mechanical configuration. This important benefit facilitates development and debug of a COP400 program prior to masking of a production part.

These devices are also economical in low and medium volume applications or when the program may require changing.

COPS and MICROBUS are trademarks of National Semiconductor Corp. TRI-STATE is a registered trademark of National Semiconductor Corp.

## Features

- Exact equivalent of the COP440/COP2440
- Socket and interface for industry standard EPROMs
- Two independent processors (COP2440)
- Dual CPU simplified task partitioning—easy to program COP2440
- Enhanced, more powerful instruction set
- 160 × 4 RAM, addresses up to 2k × 8 ROM
- MICROBUS™ compatible
- Zero-crossing detect circuitry
- True multi-vectored interrupt from four selectable sources (plus restart)
- Four level subroutine stack for each processor (in RAM)
- 4μs execution time per processor (non-overlapping)
- Single supply operation (4.5V–6.3V)
- Programmable time-base counter for real-time processing
- Software/hardware compatible with other members of COP400 family

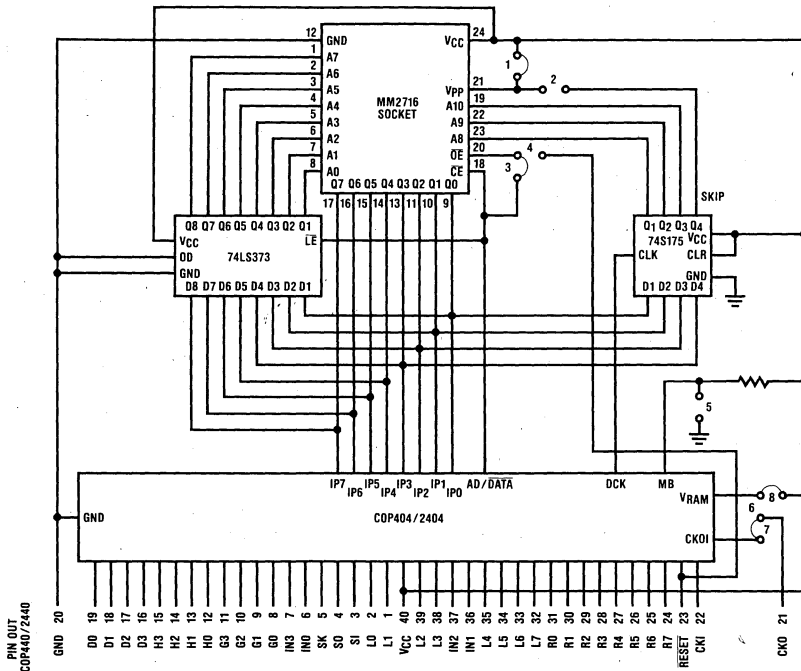
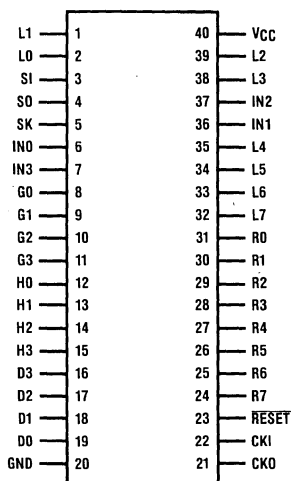
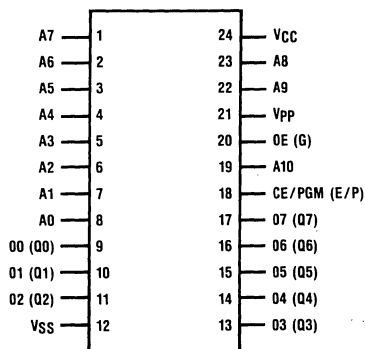


Figure 1. COP440R/COP2440R Block Diagram



COP440R/COP2440R



24-Pin Socket

Figure 2. Connection Diagrams

Pin	Description	Pin	Description
L <sub>7</sub> -L <sub>0</sub>	8-bit bidirectional I/O port with TRI-STATE®	CKI	System oscillator input
G <sub>3</sub> -G <sub>0</sub>	4-bit bidirectional I/O port	CKO	System oscillator output (or general purpose input or RAM power supply)
D <sub>3</sub> -D <sub>0</sub>	4-bit general purpose output port	<u>RESET</u>	System reset input
IN <sub>3</sub> -IN <sub>0</sub>	4-bit general purpose input port	V <sub>CC</sub>	Power supply
SI	Serial input	GND	Ground
SO	Serial output (or general purpose output)	H <sub>3</sub> -H <sub>0</sub>	4-bit bidirectional I/O port
SK	Logic-controlled clock (or general purpose output)	R <sub>7</sub> -R <sub>0</sub>	8-bit bidirectional I/O port with TRI-STATE







Section 5  
**MICROWIRE™**  
Peripherals







## COP431, COP432, COP434 and COP438 (ADC0831, ADC0832, ADC0834 and ADC0838) 8-Bit Serial I/O A/D Converters with Multiplexer Options

### General Description

The COP431 series are 8-bit successive approximation A/D converters with a serial I/O and configurable input multiplexers with up to 8 channels. The serial I/O is configured to comply with the NSC MICROWIRE™ serial data exchange standard for easy interface to the COPSTM family of processors, and can interface with standard shift registers of  $\mu$ Ps.

The 2-, 4- or 8-channel multiplexers are software configured for single-ended or differential inputs as well as channel assignment.

The differential analog voltage input allows increasing the common-mode rejection and offsetting the analog zero input voltage value. In addition, the voltage reference input can be adjusted to allow encoding any smaller analog voltage span to the full 8 bits of resolution.

### Features

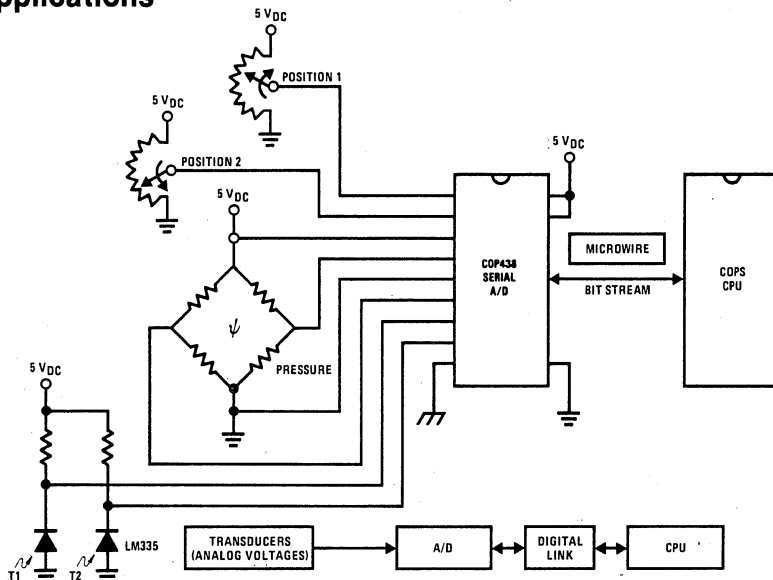
- NSC MICROWIRE compatible—direct interface to COPS family processors
- Easy interface to all microprocessors, or operates "stand-alone"

- Operates ratiometrically or with 5 V<sub>DC</sub> voltage reference
- No zero or full-scale adjust required
- 2-, 4- or 8-channel multiplexer options with address logic
- Shunt regulator allows operation with high voltage supplies
- 0V to 5V input range with single 5V power supply
- Remote operation with serial digital data link
- T<sup>2</sup>L/MOS input/output compatible
- 0.3" standard width 8-, 14- or 20-pin DIP package

### Key Specifications

■ Resolution	8 Bits
■ Total Unadjusted Error	$\pm 1/2$ LSB and $\pm 1$ LSB
■ Single Supply	5 V <sub>DC</sub>
■ Low Power	10 mW
■ Conversion Time	32 $\mu$ s

### Typical Applications



COPS™ and MICROWIRE™ are trademarks of National Semiconductor Corp.

COP431, COP432, COP434 and COP438  
(ADC0831, ADC0832, ADC0834 and ADC0838)

### Absolute Maximum Ratings (Notes 1 and 2)

Current into $V^+$ (Note 3)	10 mA
Supply Voltage, $V_{CC}$ (Note 3)	6.5V
Voltage	
Logic Inputs	- 0.3V to +18V
Analog Inputs	- 0.3V to $V_{CC} + 0.3V$
Storage Temperature	- 65°C to +150°C
Package Dissipation at $T_A = 25^\circ\text{C}$ (Board Mount)	0.8W
Lead Temperature (Soldering, 10 seconds)	300°C

### Operating Ratings (Notes 1 and 2)

Supply Voltage, $V_{CC}$	4.5 $V_{DC}$ to 6.3 $V_{DC}$
Temperature Range	0°C to 70°C

### Converter and Multiplexer Electrical Characteristics

The following specifications apply for  $V_{CC} = V^+ = 5V$ ,  $T_{MIN} \leq T_A \leq T_{MAX}$  and  $f_{CLK} = 250$  kHz unless otherwise specified.

Parameter	Conditions	Min	Typ	Max	Units
Total Unadjusted Error: (Note 4)					
ADC0831B, 32B, 34B, 38B	$V_{REF}$ Forced to 5.000 $V_{DC}$			$\pm 1/2$	LSB
ADC0831C, 32C, 34C, 38C	$V_{REF}$ Forced to 5.000 $V_{DC}$			$\pm 1$	LSB
Reference Input Resistance			2.4		k $\Omega$
Common-Mode Input Range (Note 5)	All MUX Inputs and COM Input	GND - 0.05		$V_{CC} + 0.05$	V
DC Common-Mode Error	Differential Mode		$\pm 1/16$		LSB
Power Supply Sensitivity	$V_{CC} = 5V \pm 5\%$		$\pm 1/16$		LSB
$I_{OFF}$ , Off Channel Leakage Current (Note 6)	On Channel = 5V Off Channels = 0V				
	$T_A = 25^\circ\text{C}$	- 1			$\mu\text{A}$
	On Channel = 0V Off Channels = 5V	- 50			nA
	$T_A = 25^\circ\text{C}$			1	$\mu\text{A}$
				50	nA
$I_{ON}$ , On Channel Leakage Current (Note 6)	On Channel = 0V Off Channels = 5V				
	$T_A = 25^\circ\text{C}$	- 1			$\mu\text{A}$
	On Channel = 5V Off Channels = 0V	- 200			nA
	$T_A = 25^\circ\text{C}$			1	$\mu\text{A}$
				200	nA

### AC Electrical Characteristics

The following specifications apply for  $V_{CC} = 5V$ ,  $t_r = t_f = 20$  ns and  $T_A = 25^\circ\text{C}$  unless otherwise specified.

Parameter	Conditions	Min	Typ	Max	Units
$f_{CLK}$ , Clock Frequency		10		250	kHz
Clock Duty Cycle		40		60	%
$T_C$ , Conversion Time	Not Including MUX Addressing Time			8	$1/f_{CLK}$
$t_{SETUP}$ , $\overline{SE}$ or $\overline{CS}$ Falling Edge or Data Input Valid to CLK Edge			100	250	ns

## AC Electrical Characteristics (Continued)

The following specifications apply for  $V_{CC} = 5V$ ,  $t_r = t_f = 20$  ns and  $T_A = 25^\circ C$  unless otherwise specified.

Parameter	Conditions	Min	Typ	Max	Units
$t_{HOLD}$ , Data Input Valid after CLK Rising Edge			35	90	ns
$t_{CSPW}$ , Minimum $\overline{CS}$ High Interval			35	120	ns
$t_{pd1}$ , $t_{pd0}$ —CLK Falling Edge to Output Data Valid (Note 7)	$C_L = 100$ pF Data MSB First Data LSB First		650 250	1500 600	ns ns
$t_{1H}$ , $t_{0H}$ —Rising Edge of $\overline{CS}$ to Data Output and SARS Hi-Z	$C_L = 10$ pF, $R_L = 10k$ (See TRI-STATE <sup>®</sup> Test Circuits)		125	250	ns
$C_{IN}$ , Capacitance of Logic Inputs			5		pF
$C_{OUT}$ , Capacitance of Logic Outputs			5		pF

## DC Electrical Characteristics

The following specifications apply for  $V_{CC} = 5V$  and  $T_{MIN} \leq T_A \leq T_{MAX}$  unless otherwise specified.

Parameter	Conditions	Min	Typ	Max	Units
$V_{IN(1)}$ , Logical "1" Input Voltage	$V_{CC} = 5.25V$	2.0		15	V
$V_{IN(0)}$ , Logical "0" Input Voltage	$V_{CC} = 4.75V$			0.8	V
$I_{IN(1)}$ , Logical "1" Input Current	$V_{IN} = V_{CC}$		0.005	1	$\mu A$
$I_{IN(0)}$ , Logical "0" Input Current	$V_{IN} = 0V$	-1	-0.005		$\mu A$
$V_{OUT(1)}$ , Logical "1" Output Voltage	$I_{OUT} = -360 \mu A$ , $V_{CC} = 4.75V$ $I_{OUT} = -10 \mu A$ , $V_{CC} = 4.75V$	2.4 4.5			V V
$V_{OUT(0)}$ , Logical "0" Output Voltage	$I_{OUT} = 1.6$ mA, $V_{CC} = 4.75V$			0.4	V
$I_{OUT}$ , TRI-STATE Output Current (DO, SARS)	$V_{OUT} = 0.4V$ , $T_A = 25^\circ C$ $V_{OUT} = 5V$ , $T_A = 25^\circ C$		-0.1 0.1	-100 3	$\mu A_{DC}$ $\mu A_{DC}$
$I_{SOURCE}$	$V_{OUT}$ Short to GND, $T_A = 25^\circ C$		14		mA
$I_{SINK}$	$V_{OUT}$ Short to $V_{CC}$ , $T_A = 25^\circ C$		16		mA
$I_{CC}$ , Supply Current (Note 3)			2.8		mA
$I^+$ , Current into $V^+$ (Note 3)				10	mA

**Note 1:** Absolute Maximum Ratings are those values beyond which the life of the device may be impaired.

**Note 2:** All voltages are measured with respect to ground.

**Note 3:** An internal zener diode exists from  $V_{CC}$  to GND on the  $V^+$  and  $V_{CC}$  inputs. The breakdown of these zeners is approximately 7V. The  $V^+$  zener is intended to operate as a shunt regulator and connects to the  $V_{CC}$  via a diode. When using this regulator to power the A/D, this diode guarantees the  $V_{CC}$  input to be operating below the zener voltage (7V - 0.6V). It is recommended that a series resistor be used to limit the maximum current into the  $V^+$  input.

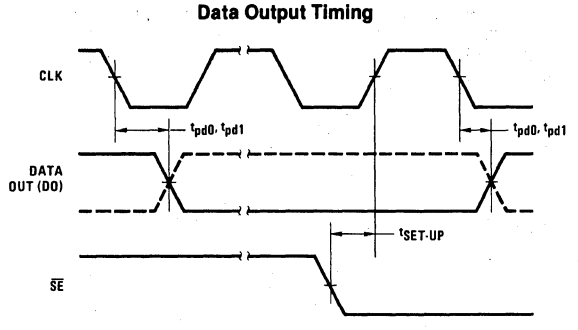
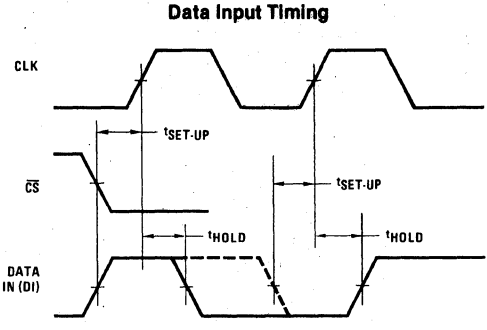
**Note 4:** Total unadjusted error includes offset, full-scale, linearity, and multiplexer errors.

**Note 5:** For  $V_{IN}(-) \geq V_{IN}(+)$  the digital output code will be 0000 0000. Two on-chip diodes are tied to each analog input (see Block Diagram) which will forward conduct for analog input voltages one diode drop below ground or one diode drop greater than the  $V_{CC}$  supply. Be careful, during testing at low  $V_{CC}$  levels (4.5V), as high level analog inputs (5V) can cause this input diode to conduct—especially at elevated temperatures, and cause errors for analog inputs near full-scale. The spec allows 50 mV forward bias of either diode. This means that as long as the analog  $V_{IN}$  does not exceed the supply voltage by more than 50 mV, the output code will be correct. To achieve an absolute 0  $V_{DC}$  to 5  $V_{DC}$  input voltage range will therefore require a minimum supply voltage of 4.950  $V_{DC}$  over temperature variations, initial tolerance and loading.

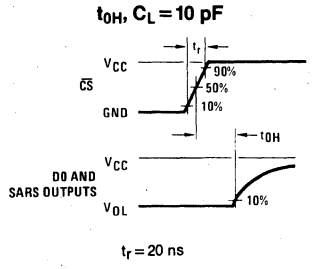
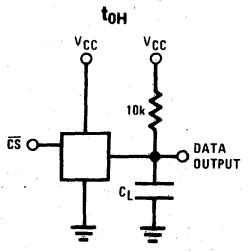
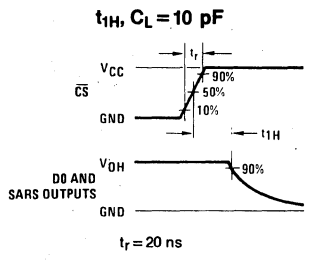
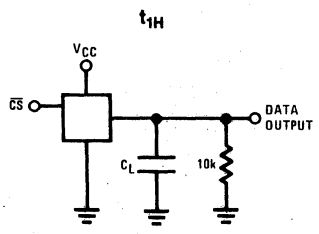
**Note 6:** Leakage current is measured with the clock not switching.

**Note 7:** Since data, MSB first, is the output of the comparator used in the successive approximation loop, an additional delay is built in (see Block Diagram) to allow for comparator response time.

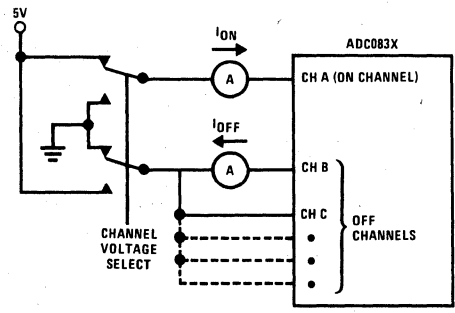
### Timing Diagrams



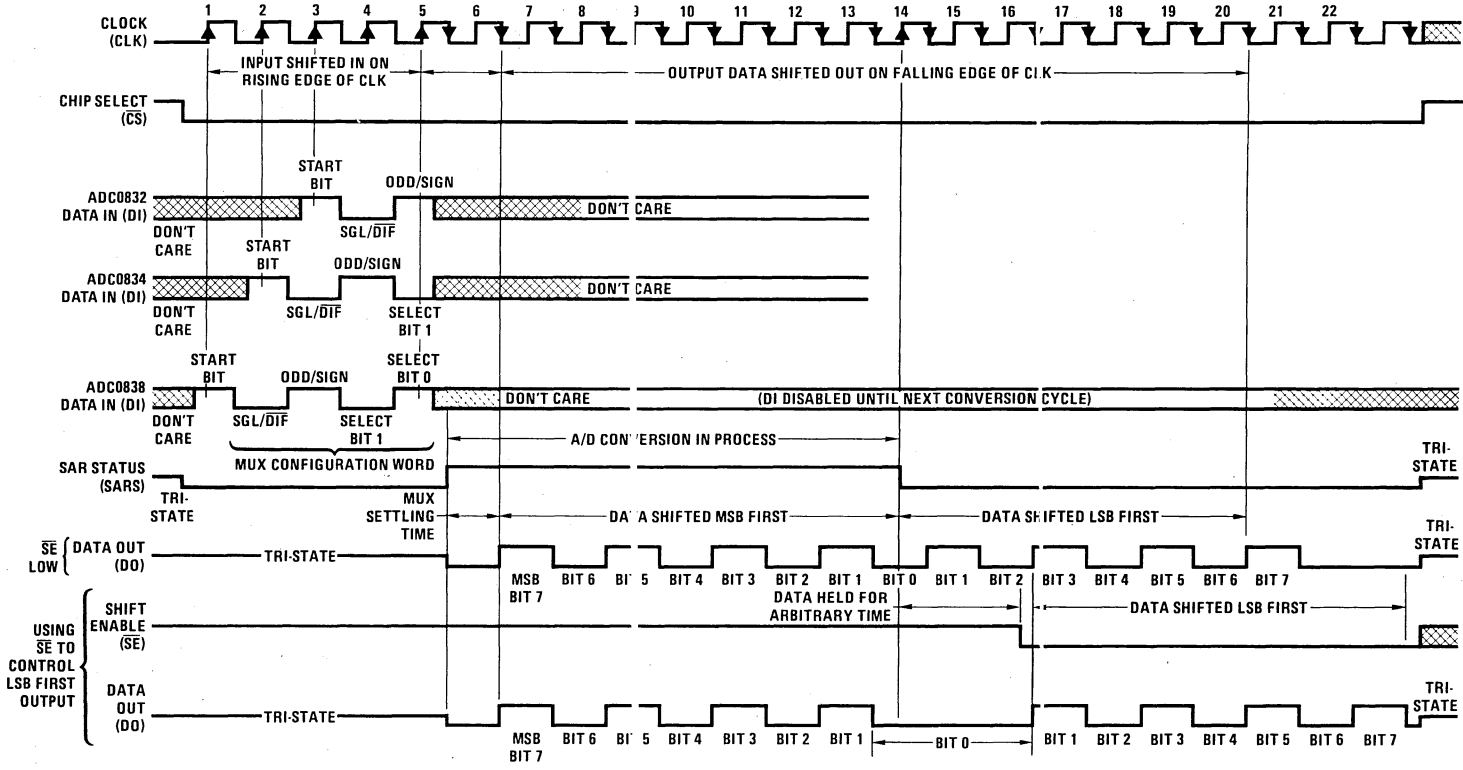
### TRI-STATE Test Circuits and Waveforms



### Leakage Current Test Circuit



# Functional Timing Diagram



COP431, COP432, COP434 and COP438  
(ADC0831, ADC0832, ADC0834 and AD(0838))



## MUX Addressing

2, 4- and 8-channel multiplexer options are available. These multiplexers are software configurable as single-ended or differential inputs. The configuration and channel assignment of the multiplexer is accomplished with a serial input word which must be preceded by a leading "1" or start bit (leading zeros are ignored).

Differential inputs are restricted to adjacent channel pairs. For example channel 0 and channel 1 may be selected as a differential pair. Channel 0 or 1 cannot act differentially with any other channel. In addition to select-

ing differential mode the sign may also be selected. Channel 0 may be selected as the positive input and channel 1 as the negative input or vice versa.

Data is always shifted in on the rising clock edge and shifted out on the falling clock edge. The only exception is the ADC0831 which requires no input data since it does not have a multiplexer. If  $\overline{CS}$  goes high, the conversion is stopped and all internal circuitry is reset. If another conversion is desired,  $\overline{CS}$  must make a high to low transition followed by address information.

TABLE I. MULTIPLEXER/PACKAGE OPTIONS

Part Number	Alternate Part Number	Number of Analog Channels		Number of Package Pins
		Single-Ended	Differential	
ADC0831	COP431	0	1	8
ADC0832	COP432	2	1	8
ADC0834	COP434	4	2	14
ADC0838	COP438	8	4	20

TABLE II. MUX ADDRESSING: ADC0838

### Single-Ended MUX Mode

MUX Address				Analog Single-Ended Channel #									
SGL/DIF	ODD/SIGN	SELECT		0	1	2	3	4	5	6	7	COM	
		1	0										
1	0	0	0	+									-
1	0	0	1			+							-
1	0	1	0					+					-
1	0	1	1								+		-
1	1	0	0		+								-
1	1	0	1				+						-
1	1	1	0						+				-
1	1	1	1									+	-

### Differential MUX Mode

MUX Address				Analog Differential Channel-Pair #							
SGL/DIF	ODD/SIGN	SELECT		0		1		2		3	
		1	0	0	1	2	3	4	5	6	7
0	0	0	0	+	-						
0	0	0	1			+	-				
0	0	1	0					+	-		
0	0	1	1							+	-
0	1	0	0	-	+						
0	1	0	1			-	+				
0	1	1	0					-	+		
0	1	1	1							-	+

# MUX Addressing (Continued)

TABLE III. MUX ADDRESSING: ADC0834

Single-Ended MUX Mode

MUX Address			Channel #			
SGL/ DIF	ODD/ SIGN	SELECT	0	1	2	3
		1				
1	0	0	+			
1	0	1			+	
1	1	0		+		
1	1	1				+

COM is internally tied to A GND

Differential MUX Mode

MUX Address			Channel #			
SGL/ DIF	ODD/ SIGN	SELECT	0	1	2	3
		1				
0	0	0	+	-		
0	0	1			+	-
0	1	0	-	+		
0	1	1			-	+

TABLE IV. MUX ADDRESSING: ADC0832

Single-Ended MUX Mode

MUX Address		Channel #	
SGL/ DIF	ODD/ SIGN	0	1
1	0	+	
1	1		+

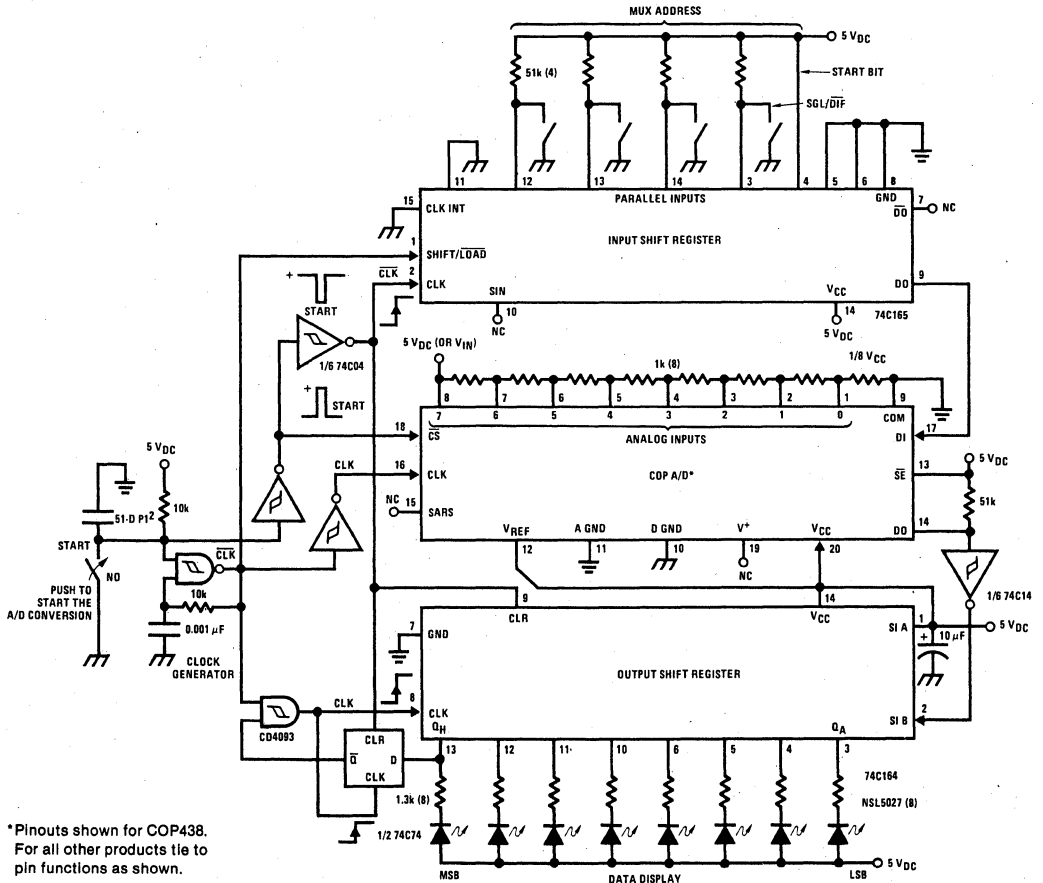
COM is internally tied to GND

Differential MUX Mode

MUX Address		Channel #	
SGL/ DIF	ODD/ SIGN	0	1
0	0	+	-
0	1	-	+

COP431, COP432, COP433, COP434 and COP438  
(ADC0831, ADC0832, ADC0833, ADC0834 and ADC0838)

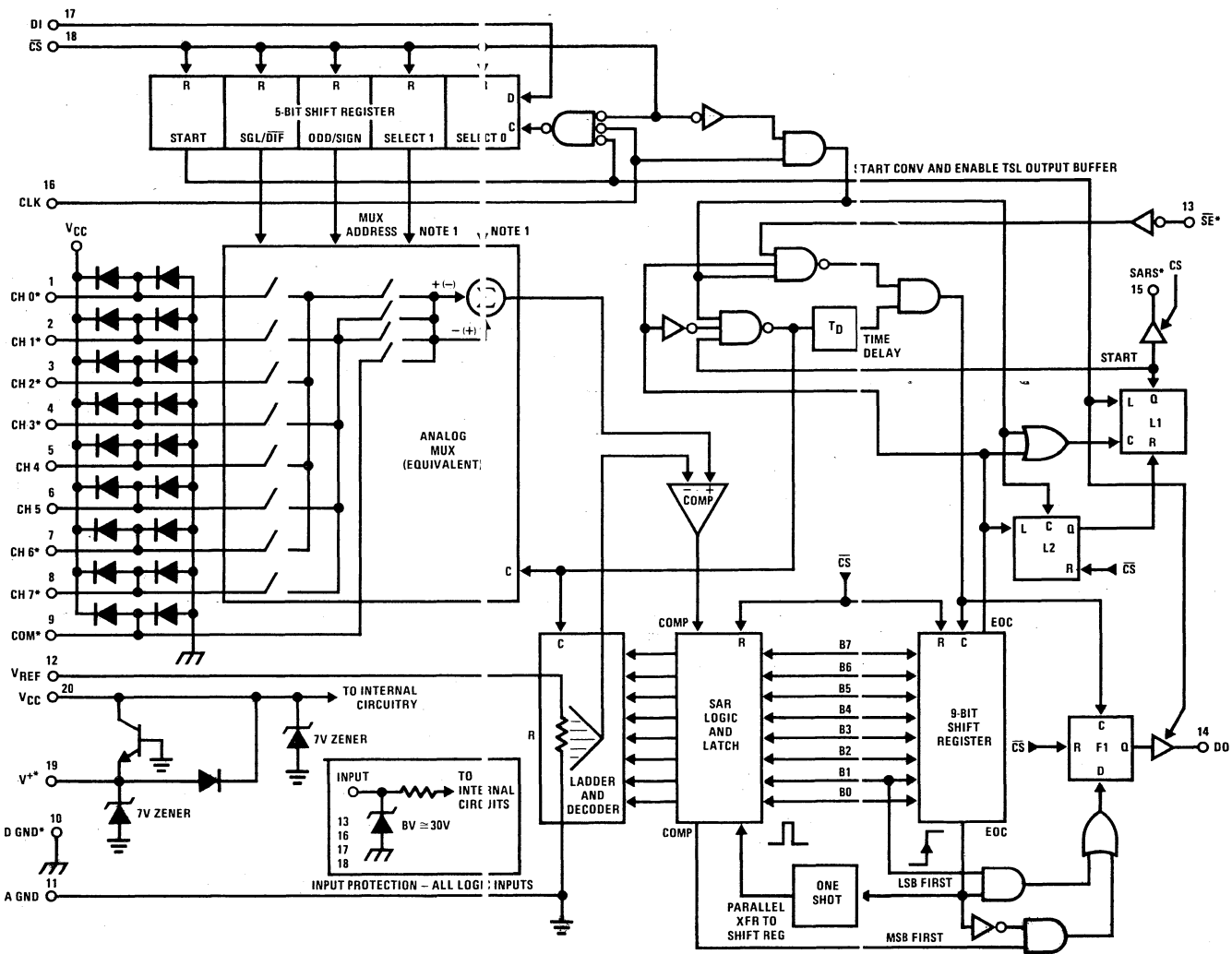
Typical Applications (Continued)



\*Pinouts shown for COP438.  
For all other products tie to  
pin functions as shown.

A "Stand-Alone" Hook-Up for COP438 Evaluation

# COP438 Functional Block Diagram



\*Some of these functions/pins are not available with other options.

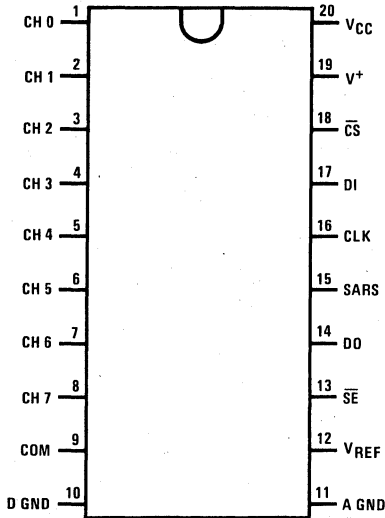
**Note 1:** For the COP434, DI is input directly to the D input of SELECT 1. SELECT 0 is forced to a "1".

COP431, COP432, COP434 and COP438  
(ADC0831, ADC0832, ADC0834 and ADC0838)

## Connection Diagrams

**COP438 8-Channel MUX**

Dual-In-Line Package

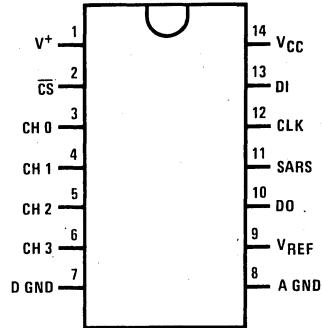


TOP VIEW

Order Number COP438BN, COP438CN  
NS Package N20A

**COP434 4-Channel MUX**

Dual-In-Line Package

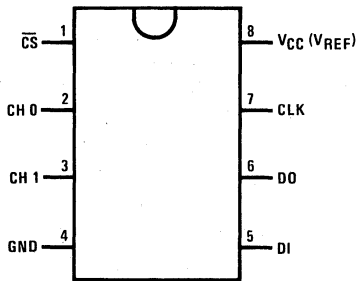


TOP VIEW

Order Number COP434BN, COP434CN  
NS Package N14A

**COP432 2-Channel MUX**

Dual-In-Line Package

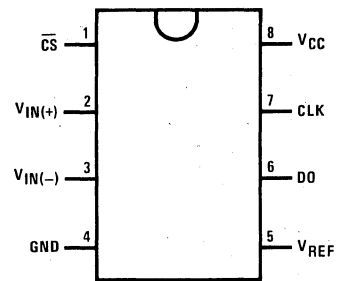


TOP VIEW

Order Number COP432BN, COP432CN  
NS Package N08A

**COP431 Single Differential Input**

Dual-In-Line Package



TOP VIEW

Order Number COP431BN, COP431CN  
NS Package N08A

## Ordering Information

# of Maximum Analog Input Channels	Linearity LSBs	Part Number	
		8	$\pm 1/2$
8	$\pm 1$	ADC0838CCN	COP438CN
4	$\pm 1/2$	ADC0834BCN	COP434BN
4	$\pm 1$	ADC0834CCN	COP434CN
2	$\pm 1/2$	ADC0832BCN	COP432BN
2	$\pm 1$	ADC0832CCN	COP432CN
1	$\pm 1/2$	ADC0831BCN	COP431BN
1	$\pm 1$	ADC0831CCN	COP431CN

# COP452/COP453 and COP352/COP353 Frequency Generator and Counter

## General Description

The COP452/COP453 and COP352/COP353 are peripheral members of the COPS™ family fabricated using N-channel silicon gate MOS technology. Containing two independent 16-bit counter/register pairs, they are well suited to a wide variety of tasks involving the measurement and/or generation of times and/or frequencies. Included in the features are multiple tone generation, precise duty cycle generation, event counting, waveform measurement, frequency bursts, delays, and "white noise" generation. An on-chip zero crossing detector can trigger a pulse with a programmed delay and duration. The COP453 is identical to the COP452, but operates with supply voltages up to 9.5 volts. The COP352/COP353 are extended temperature versions of the COP452/COP453, respectively. The COP352/COP353 are functional equivalents of the COP452/COP453.

The COP452 series peripheral devices can perform numerous functions that a microcontroller alone cannot perform. They can execute one or more complex tasks, attaining higher accuracies over a broader frequency range than a microcontroller alone. These devices remove repetitive yet demanding counting, timing, and frequency related functions from the microcontroller, thereby freeing it to perform other tasks or allowing the use of a simpler microcontroller in the system.

MICROWIRE and COPS are trademarks of National Semiconductor Corp. TRI-STATE is a registered trademark of National Semiconductor Corp.

## Features

- Unburdens microcontroller by performing "mundane" tasks
- Wider range and greater accuracy than microcontroller alone
- Generates frequencies, frequency bursts, and complex waveforms
- Measures waveform duty cycle
- Two independent pulse/event counters
- True zero crossing detector triggers output pulse
- White noise generator
- Compatible with all COP400 microcontrollers
- MICROWIRE™ compatible serial I/O
- 14-pin package
- Single supply operation  
(4.5–6.3V, COP452; 4.5–5.5V, COP352)  
(4.5–9.5V, COP453; 4.5–7.5V, COP353)
- Low cost
- Crystal or external clock  
(25 kHz to 4.44 MHz, COP452/COP453)  
(64 kHz to 4.0 MHz, COP352/COP353)
- TTL compatible

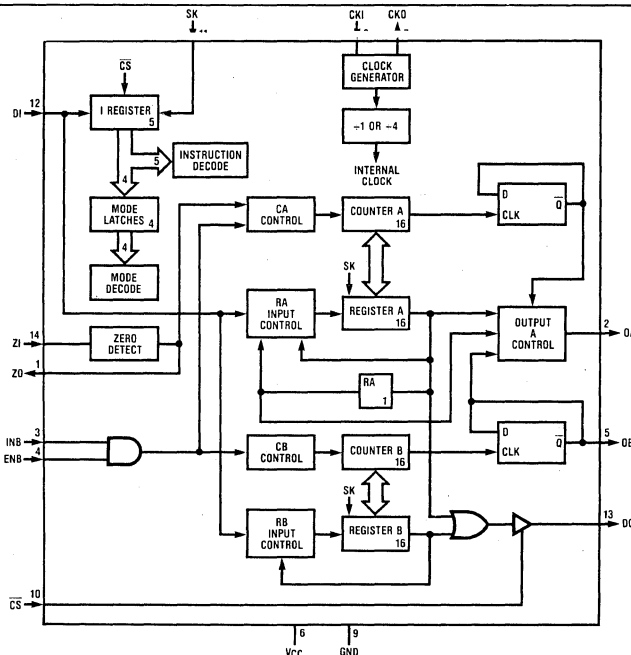


Figure 1. COP452/COP453, COP352/COP353 Block Diagram

## Absolute Maximum Ratings

Voltage at any pin (except ZI) relative to GND		Source current, outputs OA,OB	5 mA
COP452	-0.5 V to +7.0 V	Source current, all other outputs	1 mA
COP453	-0.5 V to +10 V	Total source current	10 mA
Voltage at pin ZI relative to GND	-0.8 V to +10 V	Ambient operating temperature	0°C to +70°C
Sink current, output OA	15 mA	Ambient storage temperature	-65°C to +150°C
Sink current, all other outputs	5 mA	Lead temperature (soldering, 10 sec.)	300°C
Total sink current	35 mA	Power dissipation	0.5 Watt at 25°C 0.2 Watt at 70°C

Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.

## DC Electrical Characteristics $0^{\circ}\text{C} \leq T_A \leq 70^{\circ}\text{C}$ , $4.5\text{V} \leq V_{CC} \leq 6.3\text{V}$ (COP452), $4.5\text{V} \leq V_{CC} \leq 9.5\text{V}$ (COP453) unless otherwise specified

Parameter	Conditions	Min.	Max.	Units
Operating Voltage ( $V_{CC}$ )				
COP452		4.5	6.3	V
COP453		4.5	9.5	V
Operating Supply Current	All outputs open $T_A = 0^{\circ}\text{C}$ , $V_{CC} = \text{Max.}$ $T_A = 25^{\circ}\text{C}$ , $V_{CC} = \text{Max.}$		14 12	mA mA
Input Voltage Levels				
CKI Input Levels	$V_{CC} = \text{Max.}$	3.0		V
Logic High ( $V_{IH}$ )	$V_{CC} = 5.0\text{V} \pm 5\%$	2.0		V
Logic Low ( $V_{IL}$ )			0.4	V
DI, INB, ENB, SK, CS				
Logic High	$V_{CC} = \text{Max.}$	3.0		V
Logic High ( $V_{IH}$ )	$V_{CC} = 5.0\text{V} \pm 5\%$	2.0		V
Logic Low ( $V_{IL}$ )			0.8	V
ZI Input Voltage		-0.8	+10	V
Impedance to GND at ZI		2.6	7.8	k $\Omega$
ZI Offset Voltage	(Note 1)		150	mV
Output Voltage Levels				
TTL Operation	$V_{CC} = 5.0\text{V} \pm 5\%$			
Logic High ( $V_{OH}$ )	$I_{OH} = 100\mu\text{A}$	2.4		V
Logic Low ( $V_{OL}$ )	$I_{OL} = -1.6\text{mA}$		0.4	V
Maximum Allowable Output Current Levels				
Sink Current				
OA	(Note 2)		15	mA
All Other Outputs	(Note 2)		5.0	mA
Total Sink Current	(Note 3)		35	mA
Source Current				
OA,OB	(Note 2)		-5.0	mA
All Other Outputs	(Note 2)		-1.0	mA
Total Source Current	(Note 3)		-10	mA

**Note 1:** ZI offset voltage is the absolute value of the difference between the voltage at ZI and ground (pin 9) that will cause the zero detect circuit output to change state. This is the maximum value which takes into account the worst case effects of process, temperature, voltage, and gain variation.

**Note 2:** The maximum current for the specified pin must be limited to this value or less.

**Note 3:** The total current in the device must be limited to this value or less.

**COP452/COP453****AC Electrical Characteristics**  $0^{\circ}\text{C} \leq T_A \leq 70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 6.3\text{V}$  (COP452),  $4.5\text{V} \leq V_{CC} \leq 9.5\text{V}$  (COP453)  
unless otherwise specified

Parameter	Conditions	Min.	Max.	Units
CKI Input Frequency ( $f_{IN}$ )	+4 mode	100	4440	kHz
	+1 mode	25	1110	kHz
Duty Cycle	+4	30	55	%
	+1	45	55	%
Rise Time ( $t_r$ )	$f_{IN} = 4.44\text{ MHz}$		50	ns
Fall Time ( $t_f$ )	$f_{IN} = 4.44\text{ MHz}$		40	ns
SK Input Frequency		25	250	kHz
SK Duty Cycle		30	70	%
Internal Clock Frequency ( $f_i$ )		25	1110	kHz
Internal Count Rate		0	$f_i/2$	Hz
Output Frequency		$f_i/131072$	$f_i/2$	Hz
Inputs				
DI	$t_{SETUP}$	800		ns
	$t_{HOLD}$	1.0		$\mu\text{s}$
Outputs				
CKO	$t_{pd1}$	$C_L = 50\text{ pF}$	0.2	$\mu\text{s}$
	$t_{pd0}$		0.2	$\mu\text{s}$
OA,OB	$t_{pd1}$	$C_L = 50\text{ pF}$	0.4	$\mu\text{s}$
	$t_{pd0}$		0.3	$\mu\text{s}$
ZO	$t_{pd1}$	ZI = sine wave (Figure 4)	0.7	$\mu\text{s}$
	$t_{pd0}$		0.6	$\mu\text{s}$
DO	$t_{pd1}$	$C_L = 50\text{ pF}$	1.0	$\mu\text{s}$
	$t_{pd0}$		0.6	$\mu\text{s}$



## Absolute Maximum Ratings

Voltage at any pin (except ZI) relative to GND		Source current, outputs OA,OB	5 mA
COP352	-0.5V to +7.0V	Source current, all other outputs	1 mA
COP353	-0.5V to +10V	Total source current	10 mA
Voltage at pin ZI relative to GND	-0.8V to +10V	Ambient operating temperature	-40°C to +85°C
Sink current, output OA	15 mA	Ambient storage temperature	-65°C to +150°C
Sink current, all other outputs	5 mA	Lead temperature (soldering, 10 sec.)	300°C
Total sink current	35 mA	Power dissipation	0.5 Watt at 25°C 0.125 Watt at 85°C

Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.

## DC Electrical Characteristics

-40°C ≤ T<sub>A</sub> ≤ 85°C, 4.5V ≤ V<sub>CC</sub> ≤ 5.5V (COP352), 4.5V ≤ V<sub>CC</sub> ≤ 7.5V (COP353)  
unless otherwise specified

Parameter	Conditions	Min.	Max.	Units
Operating Voltage (V <sub>CC</sub> )				
COP352		4.5	5.5	V
COP353		4.5	7.5	V
Operating Supply Current	All outputs open T <sub>A</sub> = -40°C, V <sub>CC</sub> = Max. T <sub>A</sub> = 25°C, V <sub>CC</sub> = Max.		15 12	mA mA
Input Voltage Levels				
CKI Input Levels	V <sub>CC</sub> = Max.	3.0		V
Logic High (V <sub>IH</sub> )	V <sub>CC</sub> = 5.0V ± 5%	2.2		V
Logic Low (V <sub>IL</sub> )			0.3	V
DI, INB, ENB, SK, CS				
Logic High	V <sub>CC</sub> = Max.	3.0		V
Logic High (V <sub>IH</sub> )	V <sub>CC</sub> = 5.0V ± 5%	2.2		V
Logic Low (V <sub>IL</sub> )			0.6	V
ZI Input Voltage		-0.8	+10	V
Impedance to GND at ZI		2.6	7.8	kΩ
ZI Offset Voltage	(Note 1)		150	mV
Output Voltage Levels				
TTL Operation	V <sub>CC</sub> = 5.0V ± 5%			
Logic High (V <sub>OH</sub> )	I <sub>OH</sub> = 100 μA	2.4		V
Logic Low (V <sub>OL</sub> )	I <sub>OL</sub> = -1.6 mA		0.4	V
Maximum Allowable Output Current Levels				
Sink Current				
OA	(Note 2)		15	mA
All Other Outputs	(Note 2)		5.0	mA
Total Sink Current	(Note 3)		35	mA
Source Current				
OA,OB	(Note 2)		-5.0	mA
All Other Outputs	(Note 2)		-1.0	mA
Total Source Current	(Note 3)		-10	mA

**Note 1:** ZI offset voltage is the absolute value of the difference between the voltage at ZI and ground (pin 9) that will cause the zero detect circuit output to change state. This is the maximum value which takes into account the worst case effects of process, temperature, voltage, and gain variation.

**Note 2:** The maximum current for the specified pin must be limited to this value or less.

**Note 3:** The total current in the device must be limited to this value or less.

# COP352/COP353

## AC Electrical Characteristics

$-40^{\circ}\text{C} \leq T_A \leq +85^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 5.5\text{V}$  (COP352),  $4.5\text{V} \leq V_{CC} \leq 7.5\text{V}$  (COP353)  
 unless otherwise specified

COP452/COP453, COP352/COP353

Parameter	Conditions	Min.	Max.	Units
CKI Input Frequency ( $f_{IN}$ )	+4 mode	256	4000	kHz
	+1 mode	64	1000	kHz
Duty Cycle	+4	35	55	%
	+1	50	55	%
Rise Time ( $t_r$ )	$f_{IN} = 4.0\text{MHz}$		50	ns
Fall Time ( $t_f$ )	$f_{IN} = 4.0\text{MHz}$		40	ns
SK Input Frequency		25	250	kHz
SK Duty Cycle		30	70	%
Internal Clock Frequency ( $f_i$ )		25	1000	kHz
Internal Count Rate		0	$f_i/2$	Hz
Output Frequency		$f_i/131072$	$f_i/2$	Hz
DI	$t_{SETUP}$	800		ns
	$t_{HOLD}$	1.0		$\mu\text{s}$
Outputs				
CKO	$t_{pd1}$	$C_L = 50\text{pF}$	0.25	$\mu\text{s}$
	$t_{pd0}$		0.25	$\mu\text{s}$
OA,OB	$t_{pd1}$	$C_L = 50\text{pF}$	0.45	$\mu\text{s}$
	$t_{pd0}$		0.35	$\mu\text{s}$
ZO	$t_{pd1}$	ZI = sine wave (Figure 4)	0.8	$\mu\text{s}$
	$t_{pd0}$		0.7	$\mu\text{s}$
DO	$t_{pd1}$	$C_L = 50\text{pF}$	1.1	$\mu\text{s}$
	$t_{pd0}$		0.7	$\mu\text{s}$

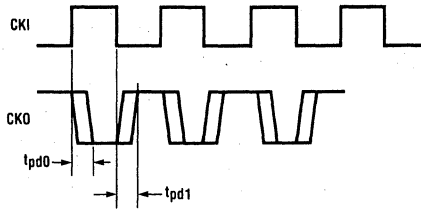


Figure 2a. CKO Output Timing

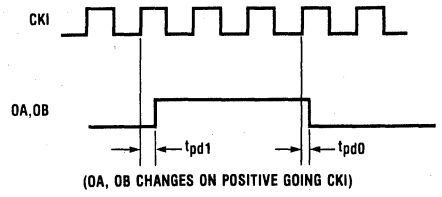


Figure 2b. OA and OB Output Timing

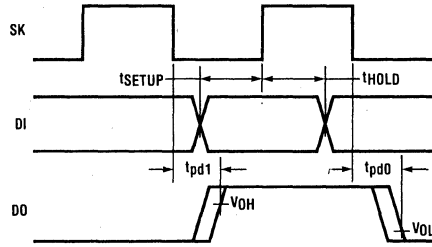


Figure 3a. Synchronous Data Timing

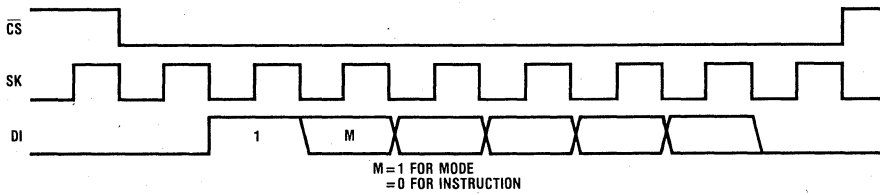


Figure 3b. Instruction Timing (Except Read/Write)

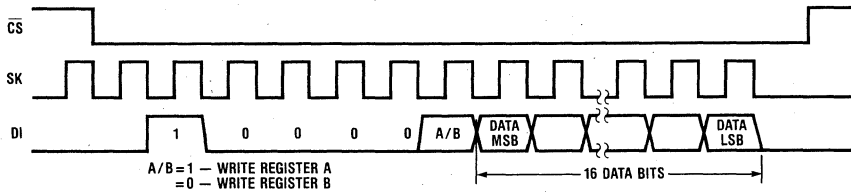


Figure 3c. Write Instruction Timing

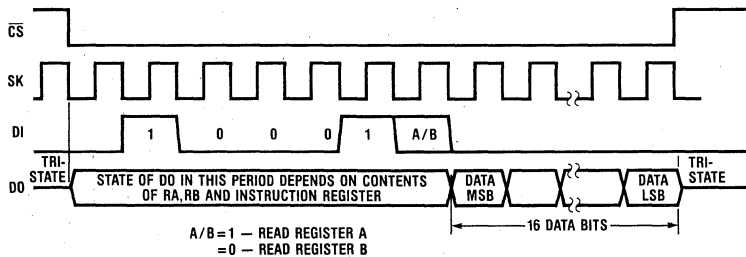


Figure 3d. Read Instruction Timing

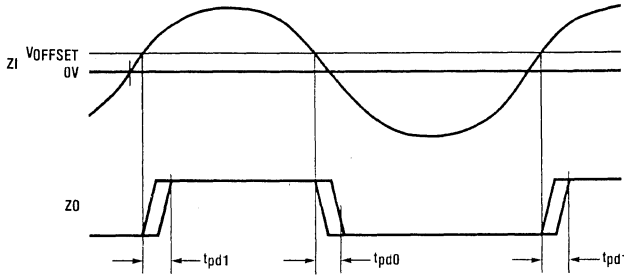


Figure 4a. ZO Timing,  $V_{OFFSET} > 0V$

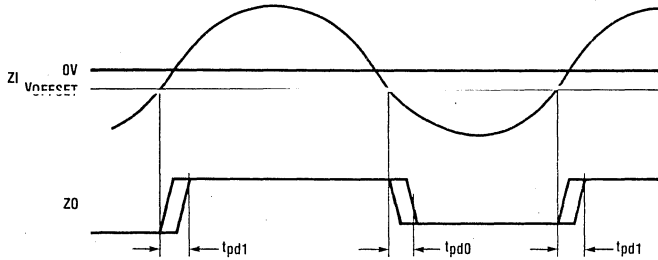
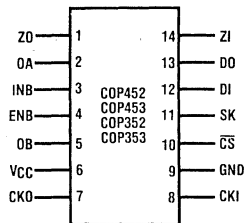


Figure 4b. ZO Timing,  $V_{OFFSET} < 0V$

Pin	Description	Pin	Description
ZO	Zero Cross Output Signal	CKI	Crystal Oscillator Input
OA	Counter A, Logic Controlled Output	GND	Ground
INB	Counter B, External Input	CS	Chip Select
ENB	Enable for INB	SK	Serial Data I/O Clock Input
OB	Counter B Output	DI	Serial Data Input
V <sub>CC</sub>	Power Supply	DO	Serial Data Output
CKO	Crystal Oscillator Output	ZI	AC Waveform Input, Counter A External Input



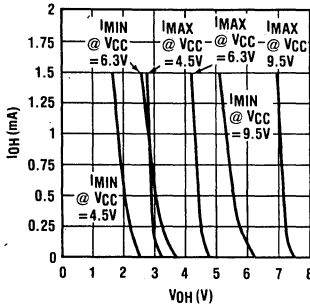
Order Number COP452N, COP352N  
NS Package N14A

Order Number COP452D, COP352D  
NS Package D14A

Figure 5. Pin Connection Diagram

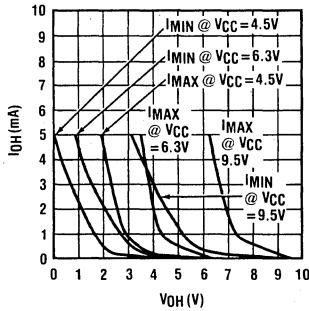
# Output Characteristics

**DO Source Current**



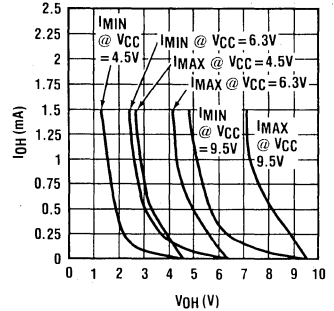
a.

**OA,OB Source Current**



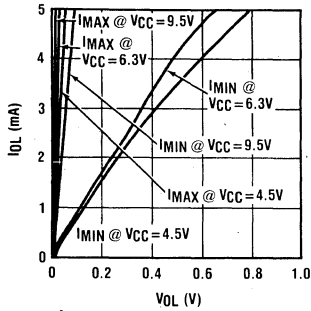
b.

**ZO Source Current**



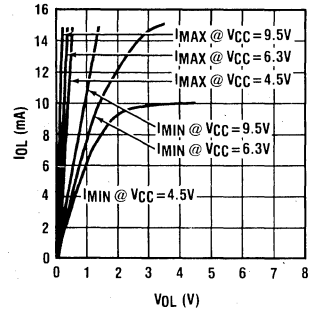
c.

**DO,ZO,OB Sink Current**



d.

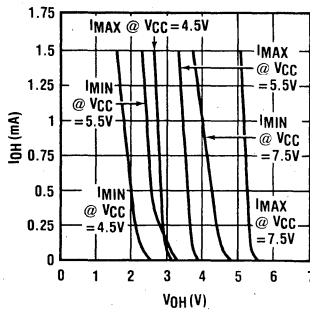
**OA Sink Current**



e.

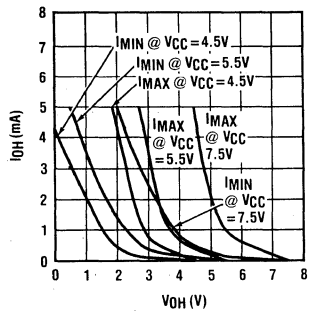
**Figure 6. COP452/COP453**

**DO Source Current**



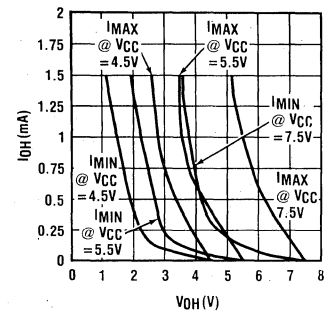
a.

**OA,OB Source Current**



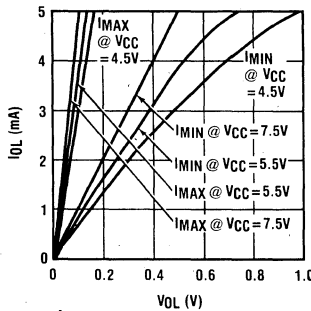
b.

**ZO Source Current**



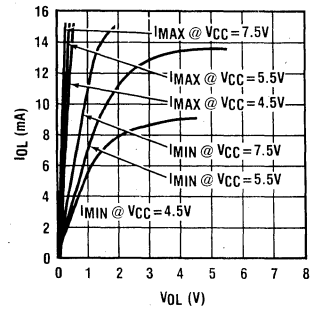
c.

**DO,ZO,OB Sink Current**



d.

**OA Sink Current**



e.

**Figure 7. COP352/COP353**

The COP452, COP453, COP352, and COP353 are functionally identical devices. They differ only in  $V_{CC}$  range and/or operating temperature range, and certain electrical parameters associated with those temperature and voltage ranges. The following information will refer only to the COP452. All the information, however, applies equally to the COP452, COP453, COP352, and COP353.

### Instruction Set and Operating Modes

The COP452 has ten instructions and eleven operating modes as indicated in Figure 8. The information for the instruction or mode is sent to the COP452 via the serial interface. The MSB is always a "1" and is properly viewed as a start bit. The second MSB identifies the communication as an instruction or a mode. The lower four bits contain the command for the device.

Instruction	Opcode		Comments
	MSB	LSB	
LDRB	100000		Load register B from DI
LDRA	100001		Load register A from DI
RDRB	100010		Read register B to DO
RDRA	100011		Read register A to DO
TRCB	100100		Transfer register B to counter B
TRCA	100101		Transfer register A to counter A
TCRB	100110		Transfer counter B to register B
TCRA	100111		Transfer counter A to register A
CK1	101000		CKI divide by one
CK4	101001		CKI divide by four
LDM	11xxxx		Load mode latches

Figure 8a. COP452 Instruction Set

Operating Mode	Opcode	
	MSB	LSB
Reset	111111	
Dual Frequency	110000	
Frequency and Count	110100	
Dual Count	110101	
Number of Pulses	110010	
Duty Cycle	110011	
Waveform Measurement	110110	
Triggered Pulse	110001	
Triggered Pulse and Count	110111	
White Noise and Frequency	111000	
Gated White Noise	111001	

Figure 8b. COP452 Operating Modes

## Functional Description

A block diagram of the COP452 is given in Figure 1. Positive logic is used. The COP452 can execute ten instructions as indicated in Figure 8a. and has eleven operating modes. The operating mode is under user software control.

The device basically consists of two sixteen bit shift registers and two sixteen bit binary down counters organized as two register-counter pairs. In most operating modes, the two register-counter pairs are completely independent of one another. For frequency generation, both the register and counter of a given pair are utilized. The counter counts down to zero where a toggle flip flop is toggled. Then the data in the register is loaded, automatically, to the counter and the process continues. A similar procedure is used in the duty cycle mode and number of pulses modes. For counting, the counters count the pulses at their respective inputs. There is no automatic counter-register transfer in the count modes. The counters wraparound from 0 to FFFF in the count modes. Data I/O is via the serial port and the registers. The counters are not involved in the input/output process at all.

The device requires a low chip select signal. When the device is selected ( $\overline{CS}$  low) the driver on the DO pin is enabled and the device will accept data at DI on each SK pulse. When the device is deselected ( $\overline{CS}$  high) the DO driver is TRI-STATE<sup>®</sup> and the I register is reset to 0. Note that chip select does not affect any other portion of the device. The mode latches are not affected. The COP452 will continue to operate in the mode specified by the user until the mode is changed by the user.

The COP452 contains a clock generator. The user may connect a crystal network to CKI and CKO or he may drive CKI from an external oscillator. Certain RC and LC networks may also be used. See the applications section for further information.

The user also has control over whether the clock generator divides the CKI signal by 4 or 1. This allows the user to quickly get a 4 to 1 change in frequency output or input count rates. Alternatively, it allows the user to use a higher speed crystal or clock generator. The internal clock frequency (the frequency after the divider) must remain between the specified limits to guarantee proper operation. The state of the divider is not affected by  $\overline{CS}$ .

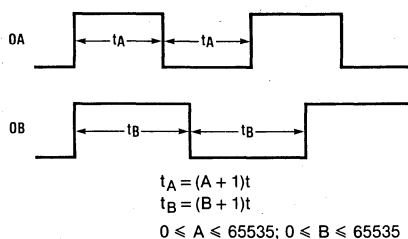
There is an internal power-on reset circuit which places the device in the Reset mode (mode latches all set to 1) and sets the clock divider to divide by four. If the CKI frequency is less than four times the minimum internal frequency the first access of the COP452 *must* be the command to set the divider to divide by 1. This command will be accepted and will be processed. Proper operation of the COP452 is not guaranteed if the internal frequency is less than the specified minimum. The power-on reset circuit does not affect the counter and registers of the COP452.

## Instruction Description

- 1. Load Register (LDRA/LDRB)** — The selected register (A/B) is loaded with 16 bits of data shifted in on DI and clocked in by SK.
- 2. Read Register (RDRA/RDRB)** — The data in the selected register (A/B) is shifted out serially onto DO. At the same time the data is recirculated back to the register.
- 3. Load Counter (TRCA/TRCB)** — The contents of the selected register are transferred to its associated counter. (Counter A is loaded from register A; counter B is loaded from register B). The contents of the register are unaffected.
- 4. Copy Counter (TCRA/TCRB)** — The contents of the selected counter are transferred to its associated register. (Counter A loads register A; counter B loads register B). The contents of the counter are unaffected.
- 5. CKI Divide by One** — The oscillator divider at the CKI input is set to divide by one. The internal frequency is therefore equal to the CKI frequency. This instruction should not be used if the CKI frequency is greater than the maximum internal frequency.
- 6. CKI Divide by Four** — The oscillator divider at the CKI input is set to divide by four. The internal frequency is therefore equal to one-fourth of the CKI frequency. This instruction should not be used if the CKI frequency is less than four times the minimum internal frequency.
- 7. Load Mode Latches** — The four mode latches are loaded with the lower four bits of the instruction.

## Mode Description

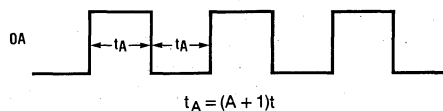
- 1. Reset Mode** — This mode sets OA and OB to "0". The mode latches are all set to "1". No counting occurs; the COP452 is in an idle condition. The registers and counters are not altered in any way.
- 2. Dual Frequency** — Two frequencies are generated — one at output OA and one at output OB. The period of the square wave at OA is determined by the contents of register A. The period of the square wave at OB is determined by the contents of register B. In frequency generation modes, the counters count down until they reach zero. At that point the output toggles and the counters are automatically loaded from the respective registers. The counters are only loaded when they count down to zero. Therefore it may be necessary to initially load the counters. The frequency outputs at OA and OB are completely independent of one another. The respective counter inputs (INB, ZI) have no effect on the counters in this mode.



Where: A = Contents of register A  
 B = Contents of register B  
 t = Period of internal clock  
 = Period of CKI oscillator (+1 mode)  
 = 4 × period of CKI oscillator (+4 mode)

Period of output square wave =  $2(N + 1)t$   
 Where t is defined above  
 N = Contents of register  
 $0 \leq N \leq 65535$  ( $0 \leq N \leq FFFF_{16}$ )

- 3. Frequency and Count** — A single frequency is output at OA. Counter B counts external pulses on INB (when ENB = 1). There is no automatic clear of the counter. Since counter B counts down from whatever state it is in it is usually desirable to preload the counter. Preloading the counter with all zeroes will give the two's complement of the count. Preloading the counter with all ones will give the one's complement of the count.



Where: A = Contents of register A  
 t = Period of internal clock  
 (as previously defined)  
 $0 \leq A \leq 65535$  ( $0 \leq A \leq FFFF_{16}$ )

OB toggles each time counter B counts through zero.  
 Maximum count rate at INB =  $f_1/2$

Where:  $f_1$  = internal clock frequency  
 = CKI input frequency (+1 mode)  
 = CKI input frequency + 4 (+4 mode)

Minimum pulse width required for reliable counting = t  
 where t = period of internal clock.

- 4. Dual Count** — In this mode counter A and counter B are enabled as external event or pulse counters. Counter A counts pulses at ZI and counter B counts pulses at INB (when ENB = 1). There is no automatic clear of either counter. Each counter counts down from whatever state it starts in. Thus, to ease reading the information, the counters should be preloaded. Preloading the counters with all zeroes will give the two's complement of the count. Preloading the counters with all ones will give the one's complement of the count. The circuitry which decrements the counters is enabled by the high to low transition at the count input. There is no interaction between the two register counter pairs.

OA toggles every time counter A counts through "0".

OB toggles every time counter B counts through "0".

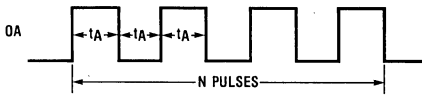
The counters, when counting, count down and wrap-around from 0 to FFFF and continue counting down.

Maximum count rate =  $f_1/2$   
 where:  $f_1$  = internal clock frequency

Minimum pulse width = t  
 where t = period of internal clock  
 (as previously defined).

There is no requirement that the count signal be symmetrical. The pulse width low must be at least equal to  $t$ . The pulse width high must also be at least equal to  $t$ .

- 5. Number of Pulses Mode** — This mode outputs at OA a specified number of pulses of a specified width. The number of pulses is specified by the contents of register B. The pulse width is specified by the contents of register A.



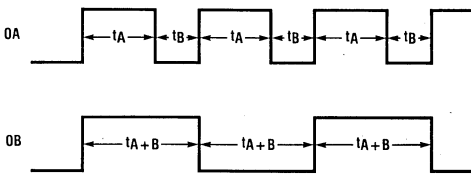
$$t_A = (A + 1)t$$

$$N = B + 1$$

Where: A = Contents of register A  
 B = Contents of register B  
 $t$  = period of internal clock (as previously defined)  
 $1 \leq A \leq 65535, A \neq 0$  ( $1 \leq A \leq FFFF_{16}$ )  
 $0 \leq B \leq 65535$  ( $0 \leq B \leq FFFF_{16}$ )

OB toggles each time a pulse train is generated at OA. The pulse train is generated each time the COP452 is selected and an instruction is sent to the device. Counter B is automatically loaded from register B after the  $N$  pulses are generated. Counter A is automatically loaded from register A at each transition of OA. Therefore simply reloading the number of pulses mode will repeat the previous sequence.

- 6. Duty Cycle Mode** — This mode generates a rectangular waveform at OA. The pulse width high is specified by the contents of register A. The pulse width low is specified by the contents of register B. A combination square wave signal is generated at OB.



$$t_A = At$$

$$t_B = Bt$$

$$t_{A+B} = (A + B)t$$

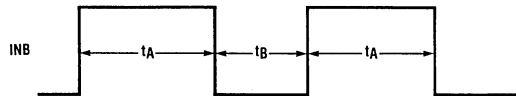
Where: A = Contents of register A  
 B = Contents of register B  
 $t$  = period of internal clock (as previously defined)  
 $1 \leq A \leq 65535, A \neq 0$  ( $1 \leq A \leq FFFF_{16}$ )  
 $1 \leq B \leq 65535, B \neq 0$  ( $1 \leq B \leq FFFF_{16}$ )

- 7. Waveform Measurement Mode** — This mode measures the high and low times of an external waveform at INB (with ENB = 1). Counter A counts the pulse width high and counter B counts the pulse width low. On the high to low transition counter A is transferred

to register A and then cleared. On the low to high transition counter B is transferred to register B and then cleared. The counters, therefore, count down from zero. Therefore the value read from the registers is a two's complement value. The transfer from the counter to register is inhibited during a read instruction.

The outputs OA and OB toggle each time the respective counter counts through zero.

The minimum pulse width, either high or low, that can be measured, is the period of the internal frequency. The maximum pulse width that can be measured is the maximum count (65535) multiplied by the period of the internal frequency.

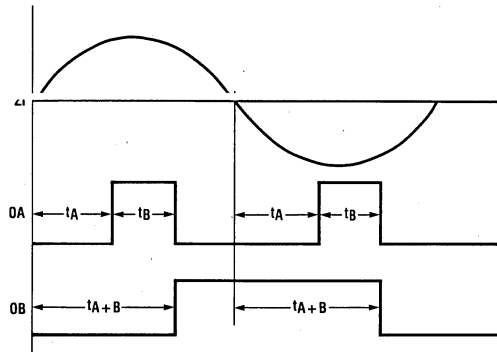


$$65535t \geq t_A \geq t$$

$$65535t \geq t_B \geq t$$

Where:  $t$  = period of internal clock

- 8. Triggered Pulse Mode** — This mode outputs a pulse triggered by the zero crossing of a signal at ZI. The delay from the zero crossing is specified by the contents of register A. The pulse width is specified by the contents of register B. Input INB is ignored. See applications section for further information.



$$t_A = (A + 1.5)t$$

$$t_B = Bt$$

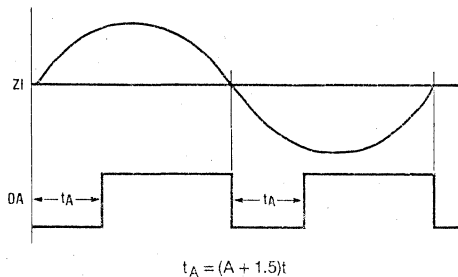
$$t_{A+B} = (A + B + 1.5)t$$

Where: A = Contents of register A  
 B = Contents of register B  
 $t$  = period of internal clock (as previously defined)  
 $0 \leq A \leq 65535$  ( $0 \leq A \leq FFFF_{16}$ )  
 $1 \leq B \leq 65535, B \neq 0$  ( $1 \leq B \leq FFFF_{16}$ )



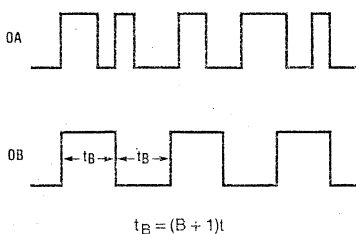
9. **Triggered Pulse and Count Mode** — This mode outputs a pulse at OA triggered by the zero crossing of a signal at ZI. The contents of register A specify the delay from the zero crossing. The pulse remains high until the next zero crossing of the signal at ZI.

Independently of the zero detection, counter B counts external events at INB (when ENB = 1). The conditions on the counter as described previously apply here.



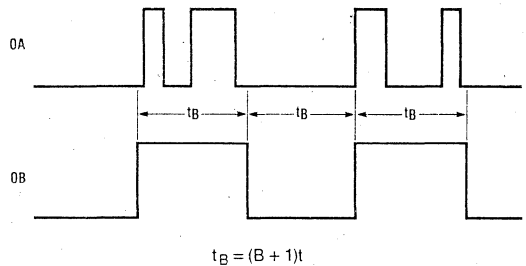
Where: A = Contents of register A  
 t = period of internal clock  
 (as previously defined)  
 $0 \leq A \leq 65535$  ( $0 \leq A \leq FFFF_{16}$ )  
 OB toggles each time counter B counts through 0

10. **White Noise and Frequency Mode** — Register A is converted to a 17-stage shift register generator for the generation of pseudo-random noise at output OA. OB outputs a square wave whose period is specified by the contents of register B. The shift register generator is shifted at the internal frequency (= CKI frequency or  $\frac{1}{4}$ CKI frequency depending on the oscillator divider). See the applications section for more information on the white noise generator.



Where: B = Contents of register B  
 t = period of internal clock  
 (as previously defined)  
 $0 \leq B \leq 65535$  ( $0 \leq B \leq FFFF_{16}$ )

11. **Gated White Noise Mode** — This mode generates pseudo-random noise ANDed with a square wave. OA outputs this combined signal. OB outputs a square wave frequency. Register A is converted into a 17-stage shift register generator which is shifted at the internal frequency rate. Counter A is not used. Counter B and register B are used in the frequency generation. See the applications section for further information on the white noise generation.



Where: B = Contents of register B  
 t = period of internal clock  
 (as previously defined)  
 $0 \leq B \leq 65535$  ( $0 \leq B \leq FFFF_{16}$ )

### General Notes

The master timing reference in the COP452 is the internal frequency. This is the CKI frequency after it has passed through the divider. This frequency must remain within its specified limits. The maximum count rate at either input is this frequency divided by 2. The minimum pulse width that can be measured is the period of this frequency.

$\overline{CS}$ , other than removing DO from the TRI-STATE<sup>®</sup> condition and allowing data to come into the I register via DI, does not affect the operation of the device.  $\overline{CS}$  must go high between accesses in order to clear the I register. Since the I register is cleared when  $\overline{CS}$  goes high, the user must insure that  $\overline{CS}$  does not go high before the COP452 has accepted the information in the I register. See the software interface section for further explanation on this point.  $\overline{CS}$  does not affect the mode latches.

In those modes where there is an automatic transfer from the register to the counter (frequency generation, duty cycle, number of pulses, triggered pulse), care must be exercised when reading or writing the register. To insure proper, "glitch-free" operation, one of the two procedures below must be followed:

1. Place the COP452 in the RESET mode.
2. Read or write the appropriate register.
3. Place the COP452 back in the original mode.

Alternatively:

1. Read or write the appropriate register.
2. Send the instruction to copy the appropriate register to its counter.

**WARNING:** Failure to observe one or the other of these procedures can cause some faulty output conditions.

The COP452 powers up in the RESET mode and with oscillator divide by 4. If the CKI input frequency is less than 4 times the minimum internal clock frequency the user *must* set the oscillator divider to divide by 1 *before* attempting any operation with the COP452. The instruction setting the oscillator divider will be accepted regardless of the value of the internal clock frequency. **Caution:** *Failure to observe this requirement will result in the improper operation of the COP452.*

## Applications Information

### Zero Cross

The ZI input normally requires a resistor and diode external to the device as indicated in Figure 9a. The resistor is part of a voltage divider used to ensure that the voltage at pin ZI does not exceed 10 volts peak and to protect the diode which is required to clamp the negative voltage swing at the input to less than  $-0.8$  volts. Figure 9b. is the recommended input circuit if logic level pulses are input to ZI for counting.

As indicated above, the input voltage at ZI must not exceed 10 volts peak. For inputs less than 10 volts peak, the resistor in Figure 9a. is required only to protect the diode. Otherwise, the resistor should be selected to guarantee that the voltage at pin ZI does not exceed 10 volts peak. Figure 10 shows this resistor ( $R_S$ ) and the impedance ( $R_{IN}$ ) which forms the first part of the input circuit at ZI. The absolute value of  $R_{IN}$  can vary widely with process variation. The user should compute the divider with  $R_S$  and the worst case maximum of  $R_{IN}$  so that the voltage at pin ZI is 10 volts or less. The following relationship should be used when the input voltage is greater than 10 volts peak:

$$\frac{R_{IN(MAX.)}}{R_S + R_{IN(MAX.)}} \times V_{IN} \leq 10 \text{ volts peak}$$

Substituting the maximum value for  $R_{IN}$  and solving for  $R_S$  gives:

$$R_S \leq \frac{V_{IN}}{10} \times 7.8k - 7.8k$$

where:  $V_{IN}$  = peak input voltage.

Note that this equation is not valid for  $V_{IN}$  less than 10 volts. In this case, the value of  $R_S$  is chosen primarily for protection of the diode and not to divide the voltage down to acceptable values.

### Zero Cross Offset

As the electrical characteristics indicates, the ZI input has a worst case offset of 150mV in the zero crossing detection. Therefore, the output of the zero cross detection circuit will change state within  $\pm 150$ mV of zero volts. There are no directional characteristics to this, i.e., approaching zero from the positive or negative direction has no effect on where the output of the zero cross detection circuit will change state (see Figure 4). The offset further indicates that the voltage at pin ZI must exceed 150mV peak in order to guarantee that the zero crossings will be detected and the appropriate signals generated.

### Triggered Pulse Modes

The delays from the zero crossing in the triggered pulse modes are measured from the point where the output of the zero crossing detection circuit changes state — the trip point of this circuit. As stated before, the delay time from this trip point is:

$$T = (A + 1.5)t$$

where: T = delay time from trip point

A = contents of register A

t = period of internal clock

The delay from the true zero crossing of the input waveform has other parameters that must be considered. The equation is of the form:

$$T = (A + 1.5)t \pm |X_1| + X_2 + X_3$$

where: T, A, t are as defined previously

$X_1$  = time for input waveform to reach the trip point of the zero cross detection circuit

$X_2$  = propagation delay through the zero cross detection circuit

$X_3$  = input synchronization delay

Parameter  $X_1$  is dependent on the peak voltage at pin ZI and on the frequency of the input signal. The peak voltage at ZI is in turn dependent on the  $R_S$ - $R_{IN}$  voltage divider and the input voltage. The  $X_1$  time is added or subtracted because the trip point of the zero cross detection circuit may be either above or below zero. In the worst case, the trip point is the maximum offset of 150mV. For a sine wave signal,  $X_1$  is determined as follows:

$$V_{OFFSET} = V_p \sin[2\pi f(X_1)]$$

$$X_1 = \frac{1}{2\pi f} \arcsin \frac{V_{OFFSET}}{V_p}$$

and

$$V_p = V_{IN} \frac{R_{IN}}{R_S + R_{IN}}$$

substituting we have

$$X_1 = \frac{1}{2\pi f} \arcsin \left( V_{OFFSET} \frac{R_S + R_{IN}}{V_{IN} R_{IN}} \right)$$

where:  $V_{OFFSET}$  = zero crossing offset or trip point

$V_p$  = peak input voltage at pin ZI

f = frequency of input signal

$R_{IN}$  = internal impedance to ground at pin ZI

$R_S$  = external series resistance at ZI

Both  $V_{OFFSET}$  and  $R_{IN}$  vary from device to device. It is clear from the equation above that the maximum value of  $|X_1|$  is obtained when  $V_{OFFSET}$  is at its maximum of 150mV and  $R_{IN}$  is at its minimum of 2.6k $\Omega$ . The minimum value of  $|X_1|$  is obtained if  $V_{OFFSET}$  is 0. Using this information, the following range of  $|X_1|$  is obtained:

$$0 \leq |X_1| \leq \frac{1}{2\pi f} \arcsin 0.15 \frac{R_S + 2.6k}{V_{IN} \times 2.6k}$$

Parameter  $X_2$  is the propagation delay through the zero crossing detection circuit and its range is given by:

$$0.3\mu s \leq X_2 \leq 0.6\mu s$$

Parameter  $X_3$  is the internal synchronization delay and is dependent upon when the zero crossing occurs relative to the internal timing which reads the output of the zero crossing detection circuit. The range for  $X_3$  is:

$$0 \leq X_3 \leq \frac{t}{2}$$

where: t = period of internal clock

With the preceding information, minimum and maximum values of the delay from true zero can be derived by simply substituting into the original equation.

$$T_{\text{MIN}} = (A + 1.5)t - \frac{1}{2\pi f} \arcsin\left(0.15 \frac{R_S + 2.6k}{V_{\text{IN}} \times 2.6k}\right) + 0.3\mu\text{s}$$

$$T_{\text{MAX}} = (A + 1.5)t + \frac{1}{2\pi f} \arcsin\left(0.15 \frac{R_S + 2.6k}{V_{\text{IN}} \times 2.6k}\right) + 0.6\mu\text{s} + \frac{t}{2}$$

The preceding information should enable the user to determine more closely the actual delay from zero of output OA of the COP452. This analysis applies to both of the triggered pulse modes. The three parameters,  $X_1$ ,  $X_2$ ,  $X_3$ , also apply in the same way in the triggered pulse and count mode when OA returns to 0 since it is the zero cross detection circuit that causes the output to return to zero in that mode.

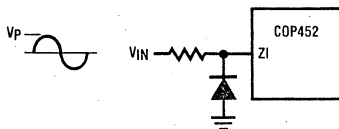


Figure 9a.

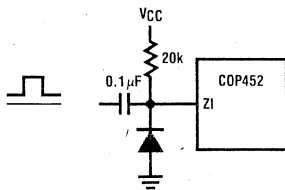


Figure 9b.

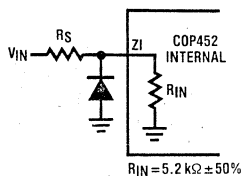


Figure 10.

### Triggered Pulse Modes: Intervening Zero Crossings

In the triggered pulse modes, it is possible to specify a delay from the zero crossing which will extend beyond the next zero crossing. In the triggered pulse and count mode, the intervening zero crossing is ignored and therefore lost. The device will still continue to operate properly. The situation is somewhat different in the "pure" triggered pulse mode where both a delay and a pulse width are specified. Any zero crossing which occurs

during the programmed delay time is ignored and therefore lost. However, if the delay time is counted out and the zero crossing occurs during the pulse width high time, the zero crossing will be recognized and the delay time will start counting again while the pulse width high time is being counted. This can result in a variety of possible conditions at the output — ranging from the apparent loss of that zero crossing to an effective very short delay from the zero crossing. What will occur depends on the values of the two counters and on their relationship to the times between zero crossings. Some interesting output waveforms can be produced, but their utility is questionable. Therefore, the user should exercise extreme caution in this mode and make sure that the times are such that all zero crossings occur at the "right" times. Otherwise, the user must be prepared to accept the bizarre effects that this situation can produce.

### Count Modes

As stated before, the counters are 16-bit down counters. Preloading them when they are enabled as external event counters with one's or zeroes will give the one's or two's complement of the count. To read the counters it is necessary to first copy the counter to its respective register and then read the register.

The user can utilize the fact that the outputs toggle when the counter counts through zero. The counter can be preloaded with a value that represents the number of events the user wishes to count. When the output corresponding to that counter toggles, the specified number of events have occurred. Thus, the user can know that the required number of events have occurred without having to actually read the counter.

The counters require a pulse width greater than or equal to the period of the internal frequency in order to be reliably decremented. It is possible for a narrower pulse to decrement the counter, but it is not guaranteed. A narrower pulse will decrement the counter if it appears at the count input at the right time relative to the internal timing of the device. Since the user does not have access to this internal timing, it is impossible for him to synchronize the count input to this timing and effectively reduce the required width of the count pulse. Therefore, applying pulses at the count input of less than one period of the internal frequency in width may cause erratic counting in the sense that some of the pulses may be recognized and some may not be recognized. Reliable counting is assured only if the width of the count pulse is greater than or equal to one period of the internal frequency.

The counters decrement on a low-going pulse at the input. As stated above, the pulse must remain low at least one internal frequency period to give reliable counting. Similarly, the count signal must go high and remain high at least one internal frequency period before it goes low again. However, the count signal does *not* have to be symmetrical.

### COP452 Oscillator

The COP452 will operate over a wide range of oscillator input frequencies. The input frequency may be supplied from an external source or CKI and CKO can be used

with a crystal or resonator to generate the oscillator frequency. Figure 11 indicates some crystal networks for some typical crystal values.

RC and LC networks can also be connected between CKI and CKO to produce the oscillation frequency. Figure 12 indicates some examples of such networks. Figure 12a. is the recommended RC network for use in this manner. With  $C_1 = 0.005 \mu\text{F}$ ,  $R = 1.5 \text{ k}\Omega$ , and  $C_2$  between  $10 \text{ pF}$  and  $400 \text{ pF}$  oscillation frequencies between about  $1 \text{ MHz}$  and  $3 \text{ MHz}$  should be obtainable. The oscillation frequency decreases with increasing values of  $C_2$ . The user should feel free to experiment with the  $R$  and  $C$  values, and with the network configuration, to produce the oscillation frequency desired.

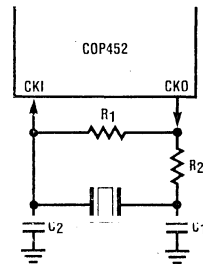
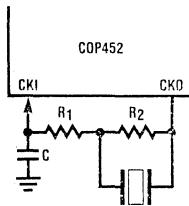
Figures 12b. and 12c. indicate LC networks that can be used to produce the COP452 oscillation frequency. In Figure 12b. with  $L = 100 \mu\text{H}$  and  $C = 100 \text{ pF}$ , a frequency of about  $2 \text{ MHz}$  should be produced. In Figure 12c., with  $L = 56 \mu\text{H}$ ,  $C_2 = 27 \text{ pF}$ , and  $C_1$  between  $25 \text{ pF}$  and  $0.01 \mu\text{F}$ , frequencies between about  $1.5 \text{ MHz}$  and  $3 \text{ MHz}$  can be produced.

There is, in effect, an inverter between CKI and CKO. This inverter was designed for use with a crystal and its associated network. It was not designed for use with

the RC and LC networks previously described. However, these networks will work and are usable. The user should be prepared to experiment with the networks to determine component values, stability, oscillation frequency, etc. These networks should be viewed as the starting point for a user who wishes to use networks of this type to generate the COP452 oscillation frequency.

The RC networks provide an inexpensive way to generate the oscillation frequency. It is foolish, however, to expect any significant degree of frequency stability or accuracy over temperature and voltage with a simple RC network — especially if inexpensive, uncompensated components are used. LC and RLC networks can produce very stable and accurate frequencies. Regardless of the network used, the user must consider the variation of the external components in his design if accuracy and stability are important considerations in his application.

The crystal networks of Figure 11 provide frequency stability and accuracy and are easy to use. If the application requires oscillation frequency accuracy and stability the crystal networks are recommended as the best solution.



Crystal Value	Component Values		
	R <sub>1</sub>	R <sub>2</sub>	C
4.44 MHz	1k	1M	27 pF
4.0 MHz	1k	1M	27 pF
3.58 MHz	1k	1M	27 pF
2.0 MHz	1k	1M	56 pF
1.0 MHz	1k	1M	56 pF

Crystal Value	Component Values			
	R <sub>1</sub>	R <sub>2</sub>	C <sub>1</sub>	C <sub>2</sub>
455 kHz	1M	16k	80 pF	80 pF
32 kHz	1M	220k	6-36 pF	30 pF

Figure 11. COP452 Crystal Oscillator

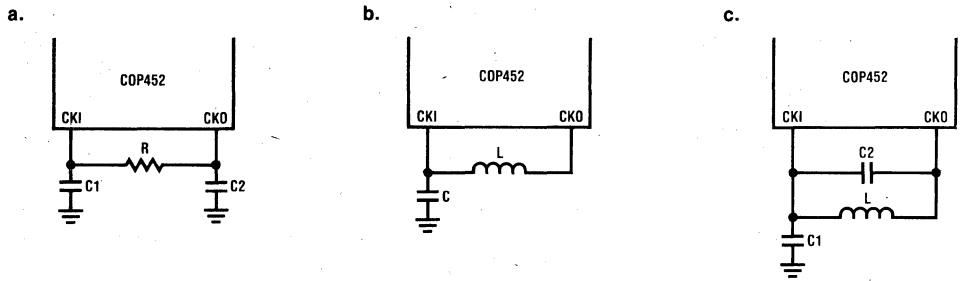


Figure 12. RC and LC Networks to Produce COP452 Oscillator Frequency

**White Noise Generation Modes**

In the two white noise modes register A is converted into a 17-stage shift register, or polynomial, generator. With feedback taps at stages 17 and 14, as indicated in Figure 13, a maximal length sequence is generated. With these feedback taps the characteristic polynomial of the sequence is:

$$X^{17} + X^3 + 1.$$

The output of this generator is a pseudo-random sequence. Since the register is shifted at the internal frequency rate, the sequence repeats after a period equal to  $(2^{17} - 1)t$ , where  $t$  is the period of the internal frequency.

The first 16 stages of the shift register are the 16 bits of register A that the user may read or write. Entering

either white noise mode presets the 16th and 17th stages to a 1 and connects the 17th stage to the shift register. If the user wishes, he can write register A and then enter the white noise and frequency mode. The output at OA will then be two "1's", and the lower 15 bits of the data user had written to register A. Following that, the polynomial sequence dictates the output. This injection of a 1 into the 16th and 17th stages prevents the lockup condition that occurs if all the stages are 0.

**Warning:** To insure proper operation, the white noise must be entered from the Reset mode. The COP452 must be in the Reset mode before the desired white noise mode and there may be no intervening modes between Reset and the desired white noise mode.

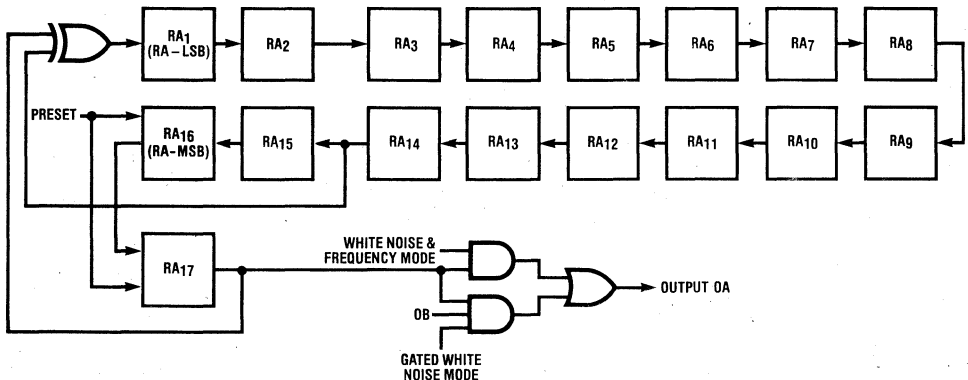


Figure 13. COP452 White Noise Generator

**Interface to COPS™ Microcontrollers**

Figure 14 indicates the typical interface between the COP452 and a COPS microcontroller. As is obvious from the figure, the interface is the standard MICROWIRE™. G<sub>2</sub> is indicated as the chip select line because it is available on all COPS microcontrollers. Obviously, any convenient output of the microcontroller may be used as the chip select for the COP452.

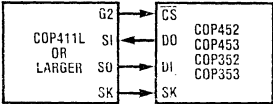


Figure 14.

The CS pin of the COP452 must be toggled between successive communications with the device. The internal i register (instruction register) is held reset (all zero) when CS is high. Since this is the only way in which the i register is cleared, failure to take CS high between accesses will result in improper operation.

The COP452 contains an internal power-on reset circuit which sets the mode latches to one, i.e., places the COP452 in the RESET mode, and sets the oscillator divider to divide by 4. The counters and registers are not affected by this reset circuit and are therefore undefined at power up.

**Interface Software for the COP452**

Sample software for interfacing COPS microcontrollers to the COP452 is given below. The code is completely general and will work in any COPS microcontroller. The following assumptions are made:

1. Pin G<sub>2</sub> is used as the chip select for the COP452 (because G<sub>2</sub> is available on all COPS microcontrollers)
2. G<sub>2</sub> is assumed high on entry to the routines.
3. The SK clock is off (0) on entry to the routines.
4. Register 0 of the microcontroller is arbitrarily chosen as the I/O register.
5. The leading digit sent out is of the form 001X where 1 is a start bit; X is 1 or 0, depending on the operation.
6. The next lower digit contains the remaining 4 bits of the command.
7. If data is being sent, it is in the next 16 bits of information sent.
8. Location GSTATE chosen as RAM address 0,15.
9. SK frequency is less than or equal to the internal frequency.

Since the COP452 is an I/O device, the code takes precautions to insure that SO is 0 prior to enabling the SK clock. (This is a wise precaution to take in any system with I/O peripherals on the serial port.)

Two version of the WRITE routine are provided. The destructive WRITE routine destroys the information in the microcontroller as the data is being sent out to the COP452. The nondestructive WRITE routine preserves the data in the microcontroller as that data is being sent out to the COP452. The destructive routine is a little more code efficient than the nondestructive routine.

```

WRCMND:   CLRA                ; SET UP POINTER FOR COMMAND ONLY WRITE
          AISC                1
          JP      WRITE
WRDATA:   CLRA                ; SET UP POINTER FOR COMMAND AND DATA WRITE
          AISC                5
WRITE:    LBI      GSTATE     ; GSTATE = LOCATION 0,15
          RMB      2
          OMG      ; SEND COP452 CHIP SELECT LOW
          CAB      ; POINT TO PROPER LOCATION FOR OUTPUT
          LEI      8          ; ENABLE SHIFT REGISTER MODE
          RC       ; JUST TO INSURE SO = 0 BEFORE CLOCK ON
          CLRA
          XAS      ; THESE 3 WORDS FOR SAFETY ONLY
          SC       ; SO SK WILL TURN ON AT NEXT XAS
SEND:     LD
          XAS
          XDS
          JP      SEND
FINISH:   RC                ; ALL DONE, SK OFF, DESELECT COP452, AND SET
          XAS      ; SO TO ZERO
DONE:     LBI      GSTATE
          SMB      2
          OMG
          LEI      0
          RET
  
```

CODE TO WRITE COP452 — DATA DESTROYED IN MICROCONTROLLER

The code below is the code to read the COP452. It is written so that the command to the COP452 is sent out nondestructively, i.e., the data in the microcontroller is preserved. A routine which sends out the data destruc-

tively could be easily generated but is not shown here. The user is referred to the techniques in the WRITE routines to determine how to modify this READ routine to send the command out destructively.

```

READ:      CLRA                ; READ INSTRUCTION IN 0, 1 AND 0, 0 AND IS
           AISC                ; OF THE FORM 00100010 OR 00100011 IF READ
           LBI      1          ; RA OR RB
           GSTATE
           RMB      2
           OMG                ; SELECT THE COP452
           CAB
           SC
           CLRA                ; SO THAT ZEROES GO OUT FIRST
           LEI      8
SEND2:     XAS
           LD
           XDS
           JP      SEND2       ; NONDESTRUCTIVE SENDING OF READ INSTRUCTION
           XAS
           CLRA                ; SET UP TO READ
           AISC      2
           CAB
           NOP                ; NOW WAIT FOR THE DATA
           NOP
           NOP
RDLOOP:    CLRA
           XAS
           XDS
           JP      RDLOOP
           RC                ; TURN OFF THE CLOCK
           XAS                ; READ LAST 4 BITS
           JP      DONE       ; COMMON EXIT WITH WRITE ROUTINE
           ; EXITS WITH DATA IN LOWER 3 DIGITS OF RO
           ; AND IN THE ACCUMULATOR

SAMPLE CODE TO READ THE COP452
WRCMND:    CLRA                ; SET UP POINTER FOR COMMAND ONLY WRITE
           AISC      1
           JP      WRITE
WRDATA:    CLRA                ; SET UP POINTER FOR COMMAND AND DATA WRITE
           AISC      5
WRITE:     LBI      GSTATE
           RMB      2
           OMG                ; SELECT THE COP452 — G2 LOW
           CAB                ; LOAD THE POINTER
           RC
           CLRA
           LEI      8          ; ENABLE SHIFT REGISTER MODE
           XAS                ; SEND OUT ZEROES
           SC
           CLRA
SEND:      XAS                ; FIRST TIME THROUGH, TURNS ON CLOCK
           LD                ; THEN SENDS DATA
           XDS
           JP      SEND
           XAS                ; SEND LAST 4 BITS
           CLRA
           NOP
FINISH:    RC
           XAS                ; ALL DONE, SK OFF
DONE:     LBI      GSTATE
           SMB      2          ; Deselect the COP452
           OMG
           LEI      0          ; SEND SO LOW
           RET

```

CODE TO WRITE COP452 — DATA PRESERVED IN MICROCONTROLLER

The software interface routines provided above are general purpose routines written to work in the general case for all COPST<sup>™</sup> microcontrollers. They are written as subroutines to be called by the main program. There is no question that other routines can be written to perform the required function. It is also clear that these routines can be reduced in specific applications. These routines should be viewed as providing a framework from which the user can develop routines which are optimal to a specific application.

Assumption 9 mentioned prior to the code itself presents an important requirement for the interface software. There must be a time delay greater than 3 periods of the internal frequency between the time the SK clock is turned off and the time the COP452 is deselected. This is required because the COP452 reads the instruction register with timing based on its internal frequency. When the microcontroller deselected the COP452,  $\overline{CS}$  goes high and the instruction register is automatically cleared. Therefore, depending on the relative speeds of SK and the internal frequency, it is possible that the instruction register may be cleared before the COP452 has accepted the information. The sample code provided automatically satisfies the requirement mentioned above whenever the SK frequency is less than or equal to the counter clock frequency. When SK is faster than the internal frequency, some delay may be required between the time SK is turned off and the time the COP452 is deselected. The time delay is not required when reading or writing the COP452 registers or when changing the oscillator divider.

**Caution:** Failure to observe this time delay will result in improper operation of the COP452.

### Application #1 — Generation of Multiple Tones

The COP452 makes the generation of two independent frequencies a simple task. This application indicates how to generate frequencies with the COP452 and also indicates other aspects or control of the device.

The requirement is to generate the following two DTMF frequencies:

$$f_1 = 941 \text{ Hz}$$

$$f_2 = 1336 \text{ Hz}$$

We will select the CKI frequency of the COP452 as 1 MHz primarily for ease in computation. Therefore, in divide by 1 mode, the internal frequency is 1 MHz. Since the registers in the COP452 are loaded with a number related to the period of the frequency, we need the periods of  $f_1$  and  $f_2$ .

$$\frac{1}{f_1} = t_1 = 1062.7 \mu\text{s}; \frac{t_1}{2} = 531.35 \mu\text{s}$$

$$\frac{1}{f_2} = t_2 = 748.5 \mu\text{s}; \frac{t_2}{2} = 374.25 \mu\text{s}$$

As stated earlier, the period of an output frequency in the COP452 in the frequency generation mode is given by:

$$T = 2(N + 1)t$$

where:  $t$  = period of internal clock  
 $N$  = register value

Solving for  $N$ , the equation becomes:

$$N = \frac{T}{2t} - 1$$

With the internal frequency at 1 MHz, the value of  $t$  is  $1 \mu\text{s}$ . Therefore, the  $N$  values with which the registers must be loaded to generate the frequencies specified above are 530 (212 hex) and 373 (175 hex). Note that the fractional parts of the numbers are lost since the COP452 cannot be loaded with fractional numbers. Note that the fractional parts may be reduced or eliminated by judicious choice of the CKI frequency. With the numbers here, the COP452 will generate a frequency with a period of  $1062 \mu\text{s}$  (941.62 Hz) and a frequency with a period of  $748 \mu\text{s}$  (1336.9 Hz). Note that these values are accurate to within 0.7% of the desired output frequencies.

Figure 15 indicates a connection diagram for this application. The software to accomplish this task is indicated below. The software indicates several aspects of the usage of the COP452. The code first resets the COP452, then loads the registers with the proper values, transfers the registers to the counters, puts the COP452 in the CKI divide by 1 state, and then loads the dual frequency mode. The output frequency generation begins when the dual frequency mode is loaded. The code as written is independent of the COP microcontroller used. The code uses the WRITE routines as described in the software interface section and assumes that these routines are located in the subroutine page.

```

PAGE          0
GSTATE       = 0, 15
POWUP:
CLRA
XAS          ; TURN OFF SK CLOCK (C = 0 AT POWER UP)
LBI         GSTATE
STII        15
LBI         GSTATE
OMG          ; MAKE SURE COP452 IS DESELECTED
LBI         0, 0
JSRP        CLEAR          ; CLEAR REGISTER 0
LBI         0, 0          ; NOW SET UP TO SEND RESET MODE TO COP452
STII        15
STII        3          ; RESET COMMAND AND START BIT
JSRP        WRCMDND

```



; THE COP452 IS NOW RESET, NOW SETUP TO WRITE REGISTER A TO  
 ; GENERATE OUTPUT FREQUENCY OF 941 HZ AT OA

```
LBI      0,0
STII    2          ; 0212 HEX = 530, GIVE PERIOD OF 1062 μs
STII    1
STII    2
STII    0
STII    1
STII    2          ; START BIT PLUS CODE TO WRITE RA
JSRP    WRDATA
```

; REGISTER A IS NOW LOADED. NEXT TRANSFER REGISTER A TO COUNTER A

```
LBI      0,0
STII    5
STII    2          ; INSTRUCTION TO TRANSFER PLUS START BIT
JSRP    WRCMND
```

; ALL DONE WITH REGISTER AND COUNTER A, NEXT WORK ON REGISTER B

```
LBI      0,0
STII    5          ; WRITE REGISTER B WITH 0175 HEX (373)
STII    7          ; TO GIVE FREQUENCY OF 1336 HZ
STII    1
STII    0
STII    0          ; INSTRUCTION TO WRITE RB
STII    2
JSRP    WRDATA
```

; REGISTER B IS NOW LOADED. NEXT TRANSFER RB TO CB

```
LBI      0,0
STII    4          ; INSTRUCTION TO TRANSFER RB TO CB
STII    2
JSRP    WRCMND
```

; NOW LOAD CKI DIVIDE BY 1

```
LIB      0,0
STII    8
STII    2
JSRP    WRCMND
```

; NOW PUT THE COP452 IN DUAL FREQUENCY MODE

```
LBI      0,0
STII    0
STII    3
JSRP    WRCMND
```

; NOW THE CODE MAY PROCEED TO DO WHATEVER ELSE IS REQUIRED IN

; THE APPLICATION.

; THE SUBROUTINES USED IN THIS APPLICATION ARE CLEAR AND THE

; WRITE ROUTINES. THE ADD ROUTINE IS USED IN THE EXAMPLE BELOW

```
        PAGE      2
```

CLEAR:

```
CLRA
XIS
JP      CLEAR
RET
```

ADD:

```
SC
LBI      2,9          ; ROUTINE ADDS 1 TO COUNTER
```

ADD1:

```
CLRA
ASC
NOP
XIS
JP      ADD1
RET
```

; WRCMND:

; SEE SOFTWARE INTERFACE FOR THIS ROUTINE

; WRDATA:

; SEE SOFTWARE INTERFACE FOR THIS ROUTINE

The preceding has done a lot with the COP452. It is clear that the code can be reduced and specialized. The purpose here was to illustrate the various communications with the device.

An interesting effect can now be produced by making use of the 4 to 1 CKI divider. With the CKI frequency at 1 MHz, the internal frequency is well within the specified

limits in either the divide by 1 or divide by 4 condition. Therefore, this characteristic of the device can be used to quickly multiply or divide the output frequency by 4. An interesting siren effect can thus be created. Sample code to do this is given below. This code assumes that the registers have been loaded and that the COP452 is in dual frequency mode. Again, the code is written to be independent of the COPS™ microcontroller used.

```

SIREN:      LBI      2, 9      ; USE REGISTER 2 AS COUNTER FOR DELAY TIME
            JSRP     CLEAR
            LBI      0, 0
            STII     8          ; CKI DIVIDE BY 1
            STII     2
            JSRP     WRCMND
PLUS1:      JSRP     ADD        ; INCREMENT COUNTER FOR DELAY
            SKC
            JP       PLUS1     ; EXIST DELAY LOOP WHEN COUNTER OVERFLOWS
            LBI      0, 0
            STII     9          ; CKI DIVIDE BY 4
            STII     2
            JSRP     WRCMND
            LBI      2, 9
            JSRP     CLEAR
PLUS1A:     JSRP     ADD
            SKC              ; AGAIN, TIME OUT VIA THE COUNTER
            JP       PLUS1A
            JP       SIREN     ; DONE, START OVER AGAIN
  
```

As is obvious from this code, it is a simple matter to create this effect. As was mentioned earlier, the code here is general purpose. This necessarily means that it can be reduced in specific applications. The user should view this code as representative of the techniques involved and then optimize or rewrite the routines to suit his particular application.

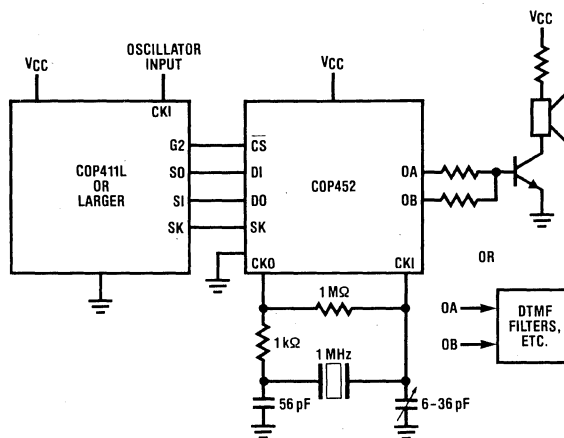


Figure 15. Dual Frequency Application

## Application #2

This application makes use of the number of pulses mode of the COP452 to control a stepping motor. The technique is equally applicable in any situation where a number of pulses must be generated based upon the state of the system. Figure 16 indicates the system interconnect. Since the oscillator frequency is 3.579545 MHz and the CKO pin of the COP452 is being used to drive the CKI of the microcontroller, a COP420 is specified as the microcontroller. If a separate oscillator were provided, any COPS™ microcontroller could be used. The software is completely general and will work in any COPS microcontroller.

The application has the following specifications:

1. The pulse width required for the stepping motor is  $5\text{ ms} \pm 5\%$ .
2. The system has 4 return lines which indicate 4 possible variations in the number of output pulses required. These four conditions are:
  - a. 10 pulses required
  - b. 100 pulses required
  - c. Repeat the last number of pulses sent
  - d. Send one more than the last number of pulses
3. The system has a signal available indicating that the return lines contain valid information.
4. One pulse is required at power up.

A flow chart to implement this system is indicated in Figure 17. Figure 16 is the interconnect used in this application. As the figure indicates, we will use a 3.579545 MHz

crystal as the time base for the COP452. With the oscillator divide by 4 selection, this gives an internal frequency period of  $1.11745\text{ }\mu\text{s}$ . With this information we can determine the number that needs to be loaded to register A to give a pulse width of 5ms. From application #1 we have the following equation which is valid here:

$$T = (N + 1)t$$

where: T = pulse width

N = contents of register A

t = period of internal clock

Solving for N we have:

$$\begin{aligned} N &= (T/t) - 1 \\ &= (5\text{ ms}/1.11745\text{ }\mu\text{s}) - 1 \\ &= 4474.34 - 1 \\ &= 4473.43 \end{aligned}$$

The fractional part is discarded, so register A must be loaded with 4473 (1179 hex) to give a 5ms pulse. The error created by the truncation of the number is  $0.5\text{ }\mu\text{s}$ . There is an error of 0.01% — well within the tolerance limits required.

The code to operate this system is given below. The interconnect of Figure 16 is assumed. The code uses the READ and WRITE subroutines as given in the software interface section of this data sheet. The code further assumes that those routines are located in the subroutine page.

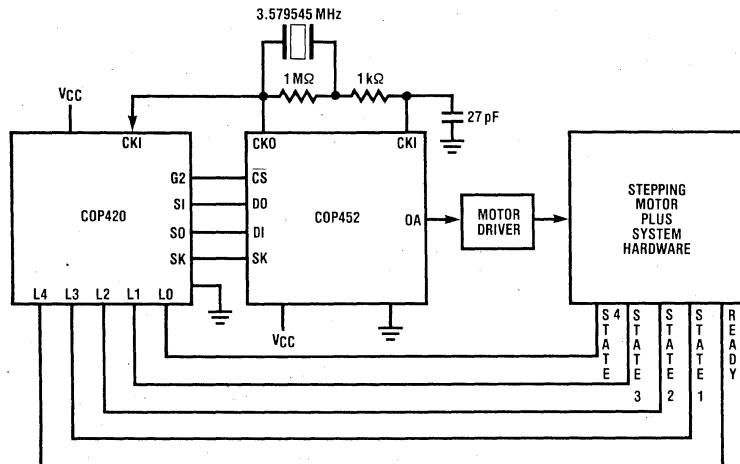


Figure 16. COP452 in Stepping Motor Control

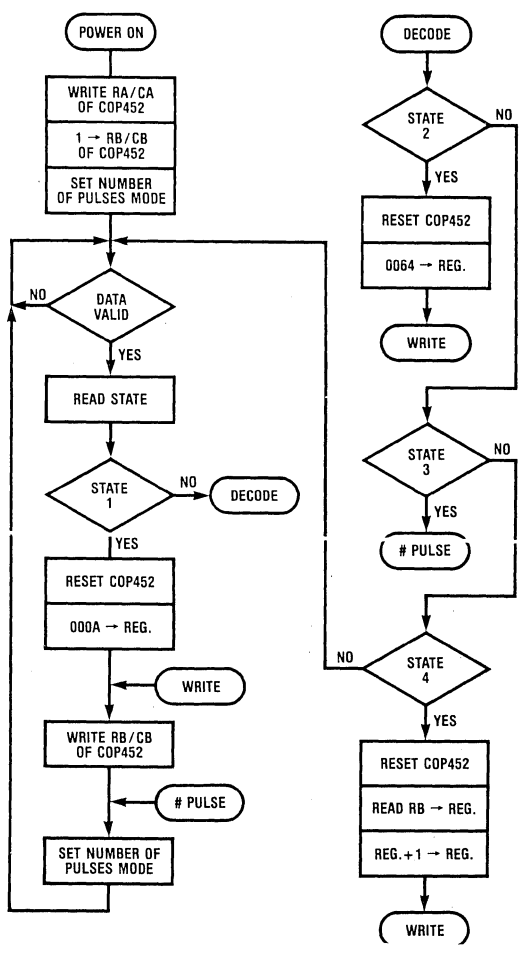


Figure 17. Flow Diagram for Application #2

```

PAGE          0
GSTATE       = 0, 15
POWRON:
CLRA
XAS          ; TURN OFF SK CLOCK
LBI         GSTATE
STII        15
LBI         GSTATE
OMG         ; DESELECT THE COP452 — G2 HIGH
LD
CAMQ        ; DRIVE THE L LINES HIGH FOR READING
LEI         4          ; ENABLE THE L OUTPUTS
LBI         0, 0
STII        9
STII        7
STII        1
STII        1
STII        1
STII        2          ; WRITE RA OF COP452 WITH 1179 HEX TO GET
JSRP        WRDATA    ; 5MS PULSE
    
```

```

LBI      0,0
STII     5      ; TRANSFER RA TO COUNTER A
STII     2
JSRP     WRCMND
LBI      0,0      ; NOW WRITE RB WITH THE NUMBER OF PULSES
STII     1
RBWRT:   STII     0      ; ONE PULSE REQUIRED AT POWER UP
RBWRT2:  STII     0
         STII     0
RBWRT3:  STII     0
         STII     2
         JSRP     WRDATA
LBI      0,0      ; NOW TRANSFER RB TO COUNTER B
STII     4
STII     2
JSRP     WRCMND
PULSE:   LBI      0,0
         STII     2      ; SET NUMBER OF PULSES MODE
         STII     3
         JSRP     WRCMND

```

```

; AT THIS POINT THE COP452 IS IN NUMBER OF PULSES MODE. ONE
; PULSE IS OUTPUT AT OA. NOW MUST READ THE RETURN LINES, MAKE
; THE APPROPRIATE DETERMINATION OF THE STATE OF THE SYSTEM
; AND UPDATE THE COP452 ACCORDINGLY. ALSO AT THIS POINT, THE
; COP452 IS SET UP TO AGAIN GENERATE A SINGLE PULSE 5MS WIDE
; IF THE DEVICE IS ACCESSED AGAIN.
;

```

```

STATE:   LBI      GSTATE
         LD        *      ; CONTENTS OF GSTATE = 15 HERE
         CAMQ      *      ; MAKE SURE L LINES ARE HIGH AND
         LEI      4      ; ENABLED
         LBI      0,0
         INL      *      ; READ THE L LINES TO A AND M(0,0)
         SKMBZ    0      ; TEST DATA — RETURN LINES — VALID
         JMP      STATE ; DATA NOT VALID, WAIT FOR IT TO BE VALID
         AISC     8      ; DATA IS VALID, DECODE A
         JMP      TEST2
STATE1:  STII     15      ; POINTING AT 0,0
         STII     3      ; RESET THE COP452 FOR STATE 1
         JSRP     WRCMND
         LBI      0,0      ; NOW SET UP TO SEND 10 PULSES
         STII     10
         JMP      RBWRT   ; SHARE COMMON CODE
TEST2:   AISC     4
         JMP      TEST3
STATE2:  STII     15      ; IN STATE2, MUST SEND 100 PULSES
         STII     3      ; FIRST RESET THE COP452
         JSRP     WRCMND
         LBI      0,0      ; WRITE 100 (0064 HEX) TO RB OF COP452
         STII     4
         STII     6
         JMP      RBWRT2
TEST3:   AISC     2
         JMP      TEST4
STATE3:  JMP      PULSE   ; STATE 3 MERELY SENDS THE SAME NUMBER OF PULSES AGAIN.
         ; THEREFORE, MERELY SEND THE NUMBER OF PULSES MODE COMMAND
         ; AGAIN

```

```

TEST4:      AISC      1
            JMP       STATE      ; ALL L LINES WERE 0, JUMP BACK TO MAIN
STATE4:     STII      15         ; RESET THE COP452
            STII      3
            JSRP      WRCMND
            LBI       0, 0       ; NOW READ THE COP452
            STII      2
            STII      2         ; COMMAND TO READ RB
            JSRP      READ
            LBI       0, 0       ; MOVE DATA TO LAST 4 DIGITS OF R0
            XIS
            XIS
            XIS
            XIS
            LBI       0, 0       ; NOW INCREMENT THE VALUE BY 1
            SC
PLUS1:      CLRA
            ASC
            NOP
            XIS
            CBA
            AISC      12
            JP        PLUS1
            JMP       RBWRT3     ; HAVE INCREMENTED THE VALUE, SEND IT OUT
;
; PAGE      2
READ:
; SEE SOFTWARE INTERFACE SECTION FOR THESE
WRDATA:    ; ROUTINES
WRCMND:

```

These are general routines and can be reduced in specific applications. The application itself was kept general so that it can be easily adapted to particular applications. The user should view this code as the basis from which to work to optimize the code for a specific application.

### Application #3

An application such as a tachometer requires the counting of external pulses that occur within a given time period. The COP452 can be used both to perform the counting and to establish the "viewing window", or time period, during which to count the pulses. By using the frequency and count mode of the COP452, a frequency can be generated which will establish this viewing time. The other counter can then be used to count the pulses. Figure 18 provides a diagram of the interconnect in this application.

As Figure 18 indicates, the oscillator frequency for the COP452 has been selected as 250 kHz. With the oscillator divider set at divide by 1, the internal frequency is also 250 kHz. At this frequency, the minimum pulse width that can be reliably expected to decrement the counter is  $4 \mu\text{s}$  — the period of the internal frequency.

A viewing time of 250 ms is arbitrarily selected. This means that the period of the output frequency is 500 ms

— a frequency of 2 Hz. Using the equation developed earlier for determining the counter values we have:

$$\begin{aligned}
 N &= \frac{T}{2t} - 1 \\
 &= (500 \text{ ms}/8 \mu\text{s}) - 1 \\
 &= 62500 - 1 \\
 N &= 62499 = \text{F423 hex}
 \end{aligned}$$

Therefore, register A must be loaded with the hex value F423 to generate a frequency of 2 Hz at OA. Counter B will count pulses when OA is high by virtue of the ENB input. When OA is low, the microcontroller will read and reset the counter and perform any necessary operations.

With the values above for the internal frequency and the viewing window, the tachometer range is 240 RPM to 62,500 RPM. By making use of the divide by 1/divide by 4 features of the oscillator divider, the range can be extended down to 60 RPM. The range when the oscillator

is divided by 4 is 60 RPM to 15,625 RPM. However, a penalty is paid for this range extension. The viewing window goes from 250ms to 1 second. The minimum reliable pulse width also increase from  $4\mu\text{s}$  to  $16\mu\text{s}$ . The added time spent counting may or may not be acceptable. It can be reduced somewhat by changing the value of RA to give a faster frequency at the reduced counter clock frequency. However, as the OA frequency increases, the low end of the range increases.

A flow chart for this application is provided in Figure 19. Sample code is given below. Note that the sample code

includes only the COP452 interface and control. Other system requirements, e.g., display interface, arithmetic, etc., are not included here. Other data sheets and application notes provide sufficient information to fill in those details.

The hardware interface indicated in Figure 18 and the code below, are completely general and valid for any COPS™ microcontroller. In specific applications both the hardware and software may be optimized to a greater extent than that shown here.

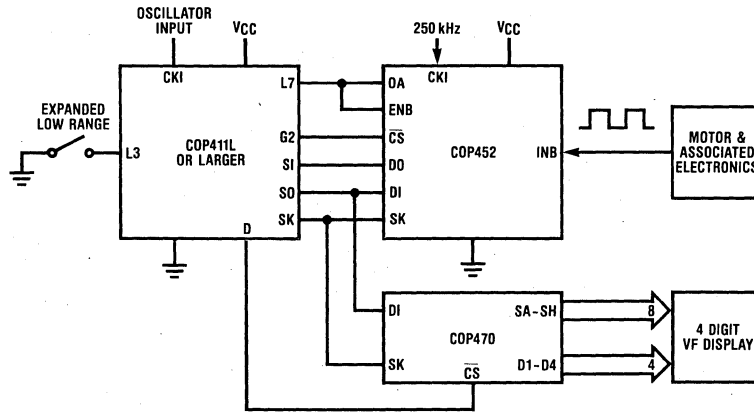


Figure 18. COP452 in Wide Range Tachometer Application

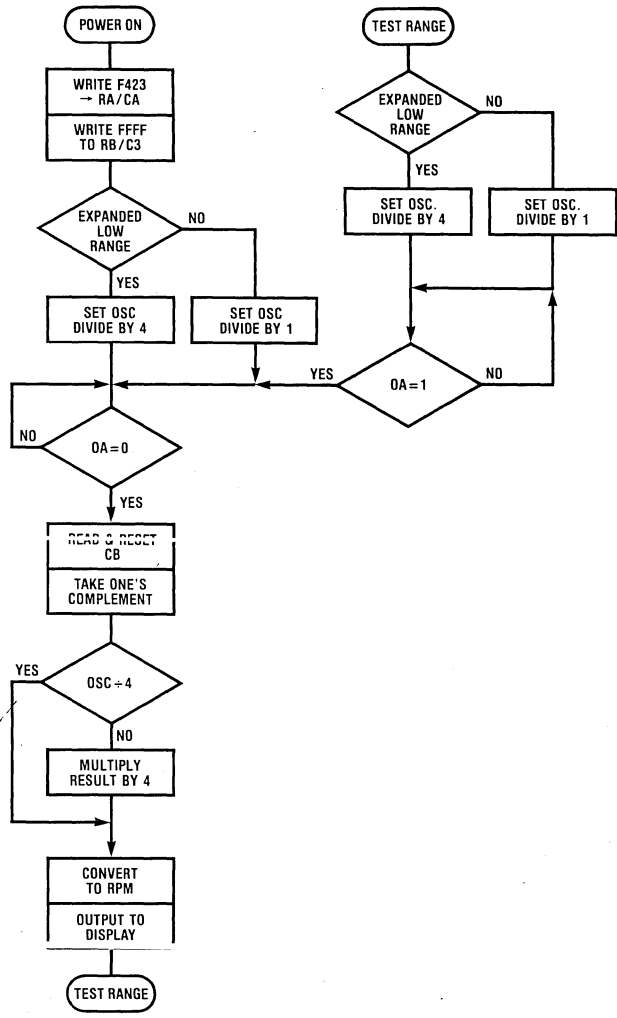


Figure 19. Flowchart for Tachometer Application

```

PAGE 0
GSTATE = 0, 15
POWRON: CLRA ; TURN OFF THE SK CLOCK—C=0 AT POWER UP
        XAS
        LBI GSTATE
        OBD ; DRIVE D LINES HIGH TO DESELECT DISPLAY
        STII 15
        LBI GSTATE
        OMG ; DESELECT THE COP452
        LD
        CAMQ ; SET THE Q REGISTER TO ALL 1'S FOR INPUT
        LBI 0, 0
        STII 3 ; NOW SET UP TO WRITE RA OF COP452
        STII 2
        STII 4
        STII 15 ; WRITE RA WITH F423 HEX
  
```



```

      STII      1
      STII      2      ; REMEMBER COP452 IS RESET AT POWER UP
      JSRP      WRDATA
      LBI       0, 0
      STII      5      ; TRANSFER RA TO CA
      STII      2
      JSRP      WRCMND
      JSR       RSTRB  ; RESET RB AND COUNTER B WITH FFFF
      JSR       RANGE  ; TEST RANGE AND SET OSCILLATOR DIVIDER
      LEI       4      ; ENABLE Q TO L—DRIVE L LINES HIGH
      LBI       0, 0  ; LOOK FOR OA = 0
TSTOA0:
      INL
      SKMBZ     3
      JP        TSTOA0
      LBI       0, 0  ; OA IS 0, READ COUNTER
      STII      6      ; FIRST TRANSFER CB TO RB
      STII      2
      JSRP      WRCMND
      LBI       0, 0  ; THEN READ RB
      STII      2
      STII      2
      JSRP      READ
      LBI       0, 0  ; NOW TAKE THE 1'S COMPLEMENT
ONECMP:
      COMP
      XIS
      COMP
      XIS
      COMP
      XIS
      COMP
      X
      LBI       0, 0  ; NOW SAVE VALUE IN R1
XFER1:
      LD        1
      XIS      1
      JP        XFER1
      JSR      RSTRB  ; RESET RB AND CB WITH FFFF FOR NEXT TIME
;
; AT THIS POINT INSERT THE APPROPRIATE CODE FOR ANY NECESSARY
; ARITHMETIC, BINARY/BCD CONVERSION, DISPLAY OUTPUT, AND ANY OTHER
; SYSTEM REQUIREMENTS. AFTER THESE ARE COMPLETE, JUMP TO LABEL
; TSTRNG WHICH HAS BEEN ARBITRARILY PLACED IN PAGE 4.
; PAGE      2
WRDATA:
;
; SEE SOFTWARE INTERFACE SECTION FOR THESE
; THREE ROUTINES
READ:
; PAGE      4
TSTRNG:
      JSR      RANGE  ; CHECK THE RANGE
      LEI     4      ; BE SURE Q IS ENABLED TO L
      LBI     0, 0  ; LOOK FOR OA = 1
TSTOA1:
      INL
      SKMBZ     3
      JMP     TSTOA0
      JP      TSTOA1
;
; THE SUBROUTINES RANGE AND RSTRB ARE INSERTED HERE
;
RANGE:
      LEI     4      ; MAKE SURE L ENABLED
      LBI     3, 15 ; WILL SAVE RANGE STATUS IN 3, 15
      INL
      X
      CLRA     ; NOW PREPARE TO SET OSCILLATOR DIVIDER

```

```

        AISC      8          ; AN 8 MEANS DIVIDE BY 1
        SKMBZ    3
        JP       HILOW
LOW:    AISC      1          ; IF DIVIDE BY 4, WANT A 9 IN A
HILOW:  LBI      0, 0
        XIS
        STII     2
        JMP      WRCMND
;
; THE FOLLOWING SUBROUTINE USES A SUBROUTINE LEVEL. IT RESETS BOTH
; REGISTER B AND COUNTER B OF THE COP452 TO FFFF
;
RSTRB:  LBI      0, 0
        STII     15
        STII     15
        STII     15
        STII     15
        STII     0
        STII     2
        JSRP     WRDATA    ; WRITE FFFF TO RB
        LBI      0, 0
        ST!!     1          ; TRANSFER RB TO CB
        STII     2
        JMP      WRCMND

```

#### Application #4

The triggered pulse mode of the COP452 provides the capability of generating the appropriate signals for triac control. Figure 20 is a general diagram of such an application.

Assume the requirement is to switch on the triac 45 degrees into the waveform. With a 60 Hz sine wave signal, the 45 degree delay is 2.0833 ms from the zero crossing. Assume also that the triac requires a gate pulse width of 150  $\mu$ s. As the diagram indicates, a 2.097 MHz crystal provides the oscillator input to the COP452. With the above information the two values that must be loaded in the COP452 can be determined. With CKI at 2.097 MHz and the oscillator divider at divide by 4, the period of the internal frequency is 1.9075  $\mu$ s. From the description of the triggered pulse mode, the pulse width is given by:

$$T = Bt$$

where: T = desired pulse width  
 B = contents of register B  
 t = period of internal clock

Solving for B is trivial and gives:

$$\begin{aligned}
 B &= T/t \\
 &= 150 \mu\text{s} / 1.9075 \mu\text{s} \\
 &= 78.64
 \end{aligned}$$

Since the register and counter can be loaded with whole numbers only, register B and counter B must be initialized with 79 (002F hex) to give a pulse width of 150  $\mu$ s.

The delay from the zero cross trip point is given by:

$$T = (A + 1.5)t$$

where: T = delay from zero cross trip point

A = contents of register A

t = period of internal clock

Solving for A we have:

$$\begin{aligned}
 A &= (T/t) - 1.5 \\
 &= (2.0833 \text{ ms} / 1.9075 \mu\text{s}) - 1.5 \\
 A &= 1090.66 \text{ rounded up to } 1091
 \end{aligned}$$

Therefore register A and counter A must be initialized with 1091 (0443 hex) to delay 2.0833 ms (45 degrees at 60 Hz) from zero cross.

Once the data has been given to the COP452 and the device placed in the triggered pulse mode, no further attention is required. The COP452 will generate the pulses with the appropriate delay as long as the power is applied and the input sine wave is available. It is a trivial matter to change any of the information. Merely write the appropriate register/counter pair. Thus very easy control is available over the firing angle of triacs.

Sample code to accomplish this function is given below. The code is general purpose and is written to work in any COPS™ microcontroller.

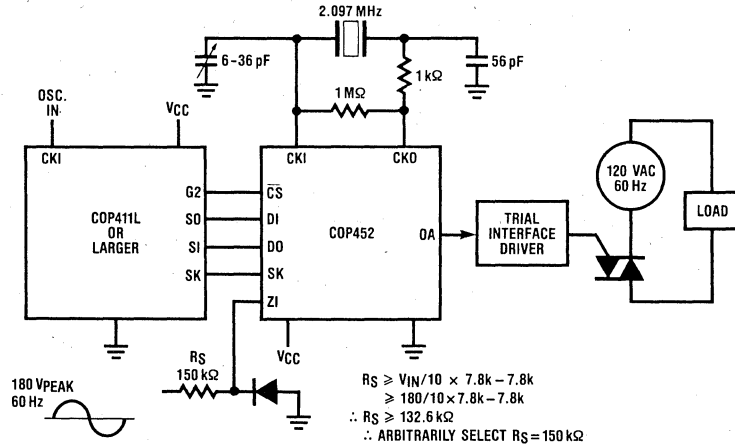


Figure 20. COP452 as Triac Controller

```

PAGE          0
GSTATE       = 0, 15
POWRON:
CLRA
XAS          ; TURN OFF THE SK CLOCK
LBI         GSTATE
STII        15
LBI         GSTATE
OMG          ; DESELECT THE COP452—G2 HIGH
LBI         0, 0 ; NOW WRITE RB/CB WITH 002F HEX TO GIVE
STII        15 ; 150µs PULSE WIDTH
STII        2
STII        0
STII        0
STII        0
STII        2
JSRP        WRDATA
LBI         0, 0
STII        4 ; TRANSFER RB TO CB
STII        2
JSRP        WRCMND
LBI         0, 0 ; NOW WRITE RA/CA WITH 0443 HEX FOR THE DELAY
STII        3
STII        4
STII        4
STII        0
STII        1
STII        2
JSRP        WRDATA
LBI         0, 0
STII        5
STII        2
JSRP        WRCMND ; TRANSFER RA TO CA
LBI         0, 0
STII        9
    
```

```

STII      2          ; SET OSCILLATOR DIVIDER TO DIVIDE BY 4
JSRP      WRCMND
LBI       0, 0
STII      1          ; SET TRIGGERED PULSE MODE
STII      3
JSRP      WRCMND

```

; ALL COMPLETE AT THIS POINT. ROUTINES WRCMND AND WRDATA ASSUMED  
; IN PAGE 2 AND ARE THE SAME AS GIVEN IN SOFTWARE INTERFACE SECTION.  
; THE COP452 WILL NOW GENERATE THE 150  $\mu$ s PULSE DELAYED BY 2.0833 ms  
; FROM EVERY ZERO CROSSING. THE USER CAN NOW IGNORE THE TRIAC CONTROL  
; AND DO WHATEVER ELSE IS REQUIRED IN THE SYSTEM. FURTHER ATTENTION  
; IS REQUIRED ONLY WHEN THE DATA IN THE COP452 MUST BE CHANGED.

Let us now compute the minimum and maximum delays from the true zero crossing in this application. As indicated earlier, the period of the internal frequency here is 1.9075  $\mu$ s. Counter A contains 0443 hex (decimal 1091).  $R_S$  is 150k and the peak input voltage is 180 volts. A 60 Hz sine wave is assumed. As given earlier, the minimum time is:

$$T_{MIN} = (A + 1.5)t - \frac{1}{2\pi f} \arcsin \left( 0.15 \frac{R_S + 2.6k}{V_{IN} \times 2.6k} \right) + 0.3 \mu s$$

Substituting we have:

$$T_{MIN} = 1092.5t - \frac{1}{120\pi} \arcsin \left( 0.15 \frac{152.6k}{180 \times 2.6k} \right) + 0.3 \mu s$$

$$= 2093.9 \mu s - 129.7 \mu s + 0.3 \mu s$$

$$T_{MIN} = 1954.5 \mu s$$

Similarly, the maximum time is given as:

$$T_{MAX} = (A + 1.5)t + \frac{1}{2\pi f} \arcsin \left( 0.15 \frac{R_S + 2.6k}{V_{IN} \times 2.6k} \right) + 0.6 \mu s + \frac{t}{2}$$

Substituting we have:

$$T_{MAX} = 1092.5t + \frac{1}{120\pi} \arcsin \left( 0.15 \frac{152.6k}{180 \times 2.6k} \right) + 0.6 \mu s +$$

$$1.9075 \mu s$$

2

$$= 2083.9 \mu s + 129.7 \mu s + 0.6 \mu s + 0.9538 \mu s$$

$$T_{MAX} = 2215.15 \mu s$$

As is obvious from the preceding analysis, the parameter previously defined as  $X_1$  is the most significant of the additional factors that define the time delay from true zero. This factor can be minimized by using as small a series resistance as possible. The frequency and input voltage will be governed by the application. The user must also remember that the minimum and maximum times calculated in this manner are absolute worst case values derived using the worst case conditions.



# COP470 and COP370 V.F. Display Driver

## General Description

The COP470 is a peripheral member of National's COPS™ Microcontroller family. It is designed to directly drive a multiplexed Vacuum Fluorescent display. Data is loaded serially and held in internal latches. The COP470 has an on-chip oscillator to multiplex four digits of eight segment display and may be cascaded and/or stacked to drive more digits, more segments, or both.

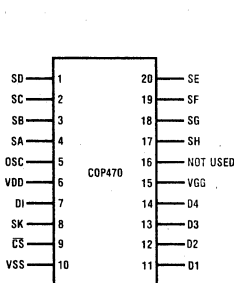
With the addition of external drivers, the COP470 also provides a convenient means of interfacing to a large-digit LED display. The COP370 is the extended temperature range version of the COP470.

## Features

- Directly interfaces to multiplexed 4 digit by 8 segment Vacuum Fluorescent displays
- Expandable to drive 8 digits and/or 16 segments
- Compatible with all COP400 processors
- Needs no refresh from processor
- Internal or external oscillator
- No "glitches" on outputs when loading data
- Drives large and small displays
- Programmable display brightness
- Small (20-pin) dual-in-line package
- Operates from 4.5V to 9.5V
- Outputs switch 35 volts and require no external resistors
- Static latches
- MICROWIRE™ compatible serial I/O
- Extended temperature device COP370 (-40°C to +85°C)

COPS and MICROWIRE are trademarks of National Semiconductor Corp.

## Connection and Block Diagrams



Order Number COP470N, COP370N  
NS Package N20A

Order Number COP470D, COP370D  
NS Package D20A

Figure 1. COP470 Pin Connection

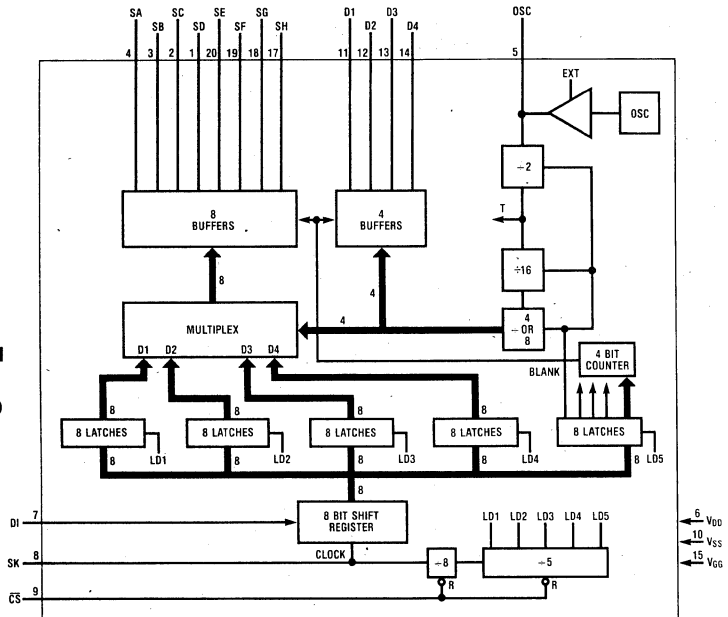


Figure 2. COP470 Block Diagram

**Absolute Maximum Ratings** ( $V_{SS} = 0$ )

Voltage at Display Outputs	+0.3V to -35V
Voltage at All Other Pins	+0.3V to -20V
Operating Temperature	
COP470	0°C to +70°C
COP370	-40°C to +85°C
Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C
Package Power Dissipation	400 mW at 25°C
	200 mW at 70°C
	125 mW at 85°C

**DC Electrical Characteristics**  $V_{SS} = 0$ ,  $V_{DD} = -4.5V$  to  $-9.5V$ ,  $V_{GG} = -30V$  to  $-35V$ ,  $T_A = 0^\circ C$  to  $70^\circ C$  for COP470 and  $T_A = 40^\circ C$  to  $85^\circ C$  for COP370 unless otherwise specified.

Parameter	Min.	Max.	Unit
Power Supply Voltage			
$V_{DD}$	-9.5	-4.5	V
$V_{GG}$ (COP470)	-35	$V_{DD}$	V
$V_{GG}$ (COP370)	-32	$V_{DD}$	V
Power Supply Current			
$I_{DD}$		5	mA
$I_{GG}$ (Display Blanked)		1	mA
Input Levels			
$V_{IH}$	-1.5	+0.3	V
$V_{IL}$	-10.0	-4.0	V
Output Drive Digits and Segments			
$I_{OH}$ @ $V_{OH} = V_{SS} - 3V$	10		mA
$I_{OH}$ @ $V_{OH} = V_{SS} - 2V$	7		mA
$I_{OL}$ @ $V_{OL} = V_{GG} + 2V$ (See Note 1.)	10		$\mu A$
Output Drive @ $V_{GG} = V_{DD} = V_{SS} - 5V$			
$I_{OH}$ @ $V_{OH} = V_{SS} - 2V$	1		mA
Allowable Source Current			
Per Pin		20	mA
total for segments		$\infty$	mA
Input Capacitance		7	pF
Input Leakage		1	$\mu A$

**AC Electrical Characteristics**

OSC Period (internal or external)	4	20	$\mu s$
OSC Pulse Width	1.5		$\mu s$
Clock Period T (twice Osc. period)	8	40	$\mu s$
Display Frequency			
4 digits = $1/64T$	390	2000	Hz
8 digits = $1/128T$	190	1000	Hz
SK Clock Frequency	0	250	kHz
SK Clock Width	1.5		$\mu s$
Data Set-up and Hold Time			
$t_{SETUP}$	1.0		$\mu s$
$t_{HOLD}$	50		ns
CS Set-up and Hold Time			
$t_{SETUP}$	1.0		$\mu s$
$t_{HOLD}$	1.0		$\mu s$
Duty Cycle			
4 digits	1/64	15/64	
8 digits	1/128	15/128	

**Note 1.**  $I_{OL}$  current is to  $V_{GG}$  with the chip running. Current is measured just after the output makes a high-to-low transition.

## Timing Diagram

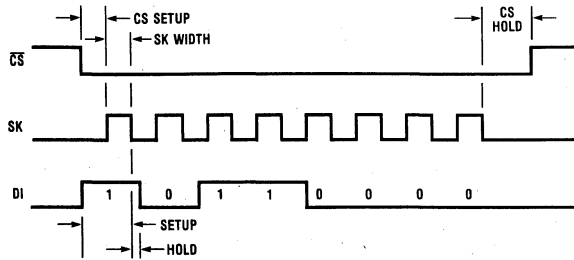
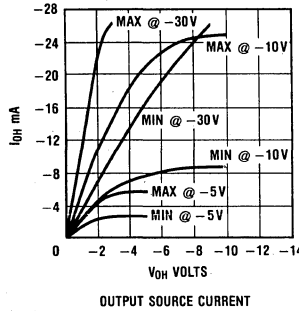


Figure 3. Serial Load Timing Diagram

## Performance Characteristic



## Functional Description

### Segment Data Bits

Data is loaded in serially in sets. Each set of segment data is in the following format:

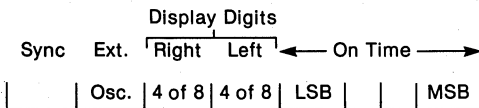


Data is shifted into an eight bit shift register. The first bit of the data is for segment H, digit 1. The eighth bit is segment A, digit 1.

A set of eight bits is shifted in and then loaded into the digit one latches. The second set of 8 bits is loaded into digit two latches. The third set into digit three latches and the fourth set is loaded into digit four latches.

### Display on Time and Control Bits

The fifth set of 8 data bits contains blank time data and control data in the following format:



the first four bits shifted in contain the on time. This is used to control display brightness. The brightness is a function of the on time of each segment divided by the total time (duty cycle). The on time is programmable from 0 to 15 and the total time is 64. For example, if the on time is 15, the duty cycle is 15/64 which is maximum brightness. If on time is 8, the duty cycle is 8/64, about 1/2 brightness. There are 16 levels of brightness from 15/64 to 0/64 (off).

The fifth and sixth bits control the multiplex digits. To enable the COP470 to drive a 4 digit multiplex display, set both bits to one. If two COP470s are used to drive an 8 digit display, bit five is set on the left COP470 and bit six is set on the right COP470 (see Fig. 6). In the eight digit mode, the display duty cycle is on time/128.

The seventh bit selects internal or external oscillator. The OSC pin of the COP470 is either an output of the internal oscillator (bit 7 = 0) or is an input allowing the COP470 to run from an external oscillator (bit 7 = 1).

The eighth bit is set to synchronize two COP470s. For example, to set the COP470 to internal osc, 4 digits, and maximum brightness, send out six ones and two zeros.

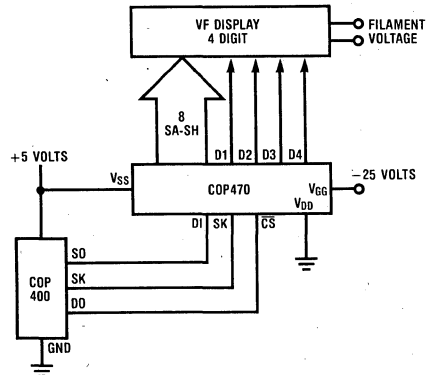


Figure 4. System Diagram — 4 Digit Display

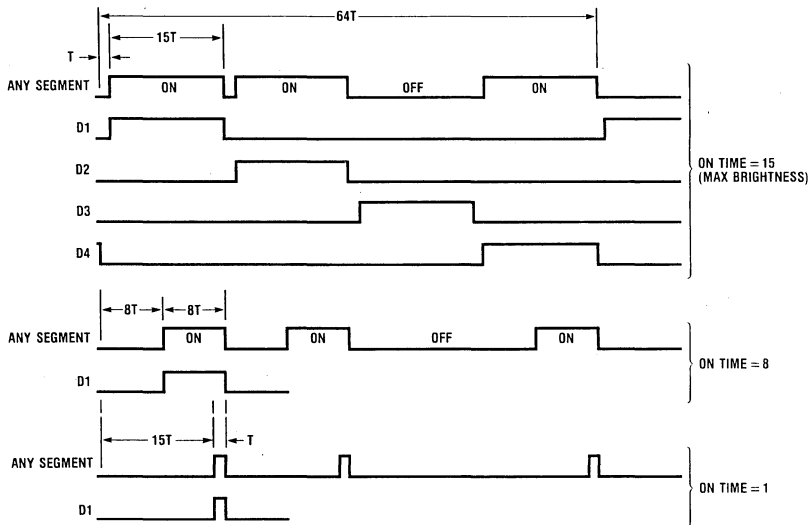


Figure 5. Segment and Digit Output Timing Diagram

### Loading Sequence:

#### Step

- 1 Turn  $\overline{CS}$  Low.
- 2 Clock in 8 bits of data for digit 1.
- 3 Clock in 8 bits of data for digit 2.
- 4 Clock in 8 bits of data for digit 3.
- 5 Clock in 8 bits of data for digit 4.
- 6 Clock in 8 bits of data for on time and control bits.
- 7 Turn  $\overline{CS}$  high.

Note:  $\overline{CS}$  may be turned high after any step. For example, to load only 2 digits of data do steps 1, 2, 3, and 7.  $\overline{CS}$  must make a high to low transition before loading data in order to reset internal counters.

### 8 Digit Displays

Two COP470s may be tied together in order to drive an eight digit multiplexed display. This is shown in Figure 6. The following is the loading sequence to drive an eight digit display using two COP470s.

1. Turn  $\overline{CS}$  low on both COP470s.
2. Shift in 32 bits of data for the right 4 digits.

3. Shift in 4 bits of on time, a zero and three ones. This synchronizes both chips, sets to external oscillator, and to right four of eight digits. Thus both chips are synchronized and the oscillator is stopped.
4. Turn  $\overline{CS}$  high to both chips.
5. Turn  $\overline{CS}$  low to the left COP470.
6. Shift in 32 bits of data for the left 4 digits.
7. Shift in 4 bits of on time, a one and three zeros. This sets this COP470 to internal oscillator and to left four of eight digits. Now both chips start and run off the same oscillator.
8. Turn  $\overline{CS}$  high.

The chips are now synchronized and driving eight digits of display. To load new data simply load each chip separately in the normal manner.

### 16 Segment Display

Two COP470s may be tied together in order to drive a sixteen segment display. This is shown in Figure 8. To do this, both chips must be synchronized, one must run off external oscillator while the other runs off its internal oscillator outputting to the other. Similarly, four COP470s could be tied together to drive eight digits of sixteen segments.



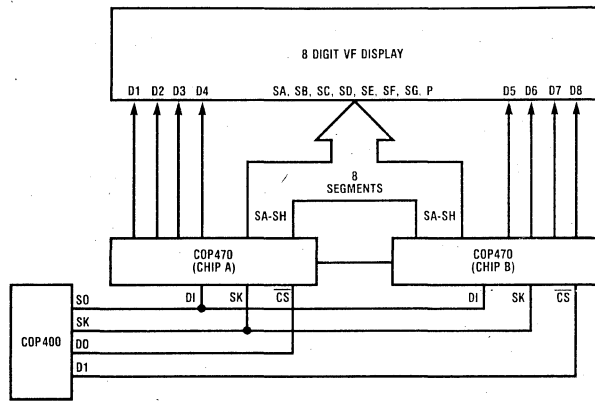


Figure 6. System Diagram 8 Digit Display

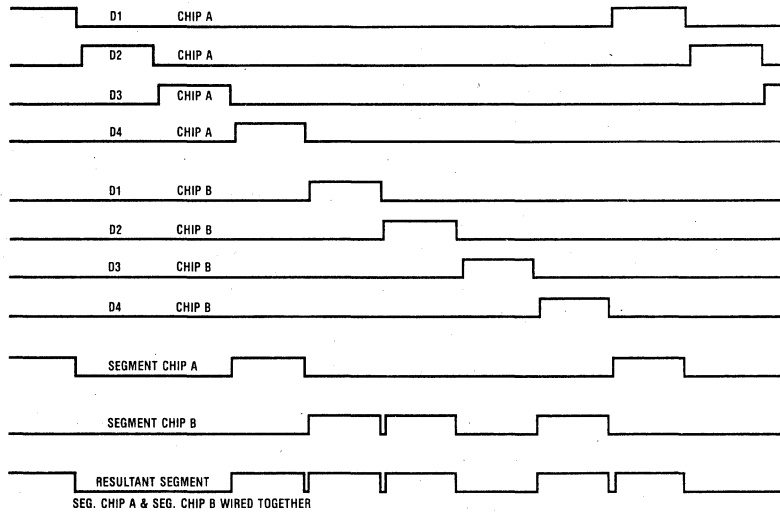


Figure 7. Segment and Digit Output Timing Diagram for 8 Digits

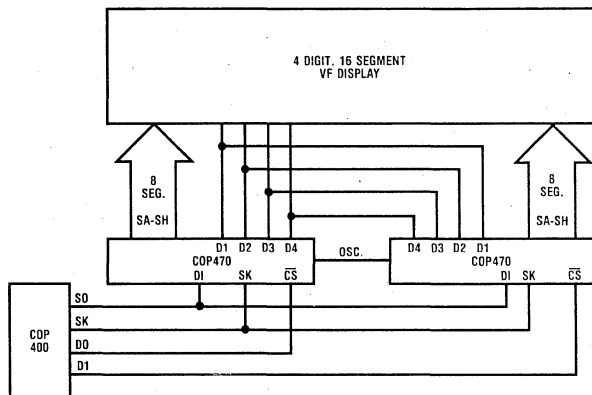


Figure 8. System Diagram for 16 Segment Display

## LED Display

The COP470 may be used to drive LED displays. The COP470 can drive the segments directly on small, low current LED displays as shown in Figure 9. By adding

display drivers, large, high current LED displays can be driven as shown in Figure 10.

### Example:

#### COP420 Code to Load COP470

(Display Data is in Memory 0, 12 — 0, 15)

```

                                ; Point to first display data
                                ; Turn CS low (DO)
LOOP:  LBI 0,12
                                ; Look up segment data
                                ; Copy data from Q to M & A
                                ; Set C to turn on SK
                                ; Output lower 4 bits of data
                                ; Delay
                                ; Delay
                                ; Load A with upper 4 bits
                                ; Output 4 bits of data
                                ; Delay
                                ; Delay
                                ; Reset C
                                ; Turn off SK clock
                                ; Increment B for next data
                                ; Skip this jump after last digit
                                ; Set C
                                ;
                                ; 15 to A
                                ; Output on time (max brightness)
                                ;
                                ;
                                ; 12 to A
                                ; Output control bits
                                ;
                                ; 15 to B
                                ; Reset C
                                ; Turn off SK
                                ; Turn CS high (DO)

```

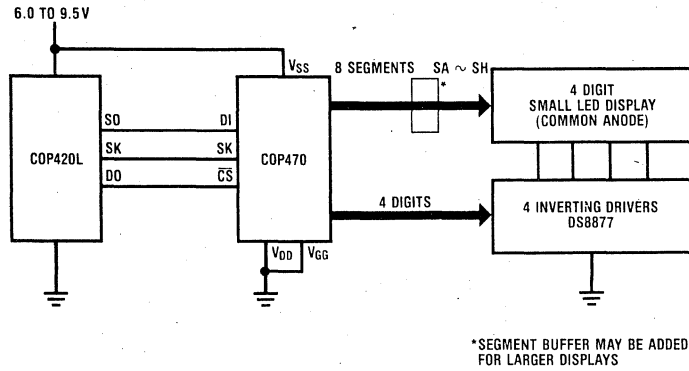


Figure 9. LED Display

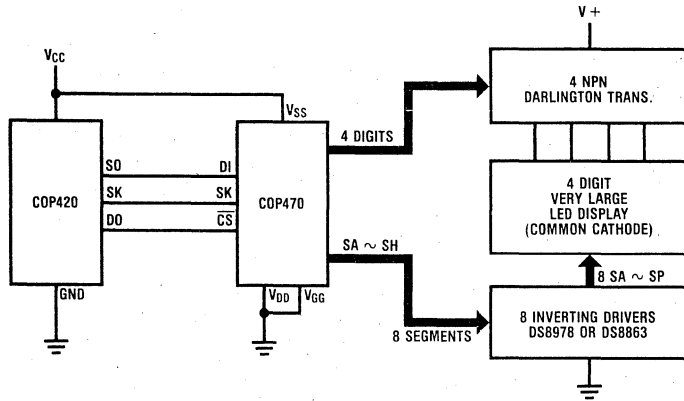


Figure 10. Large LED Display

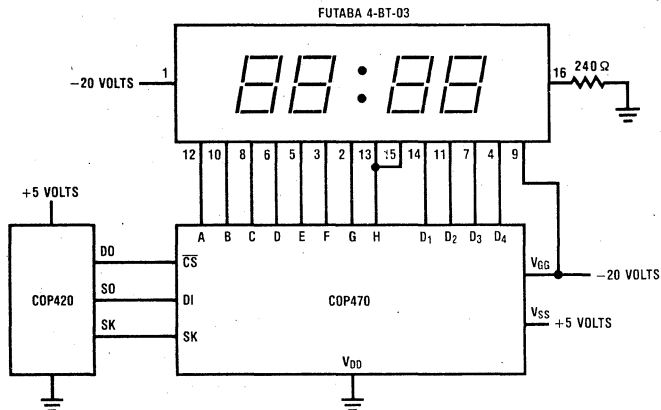


Figure 11. Sample V.F. System

## COP472 Liquid Crystal Display Controller

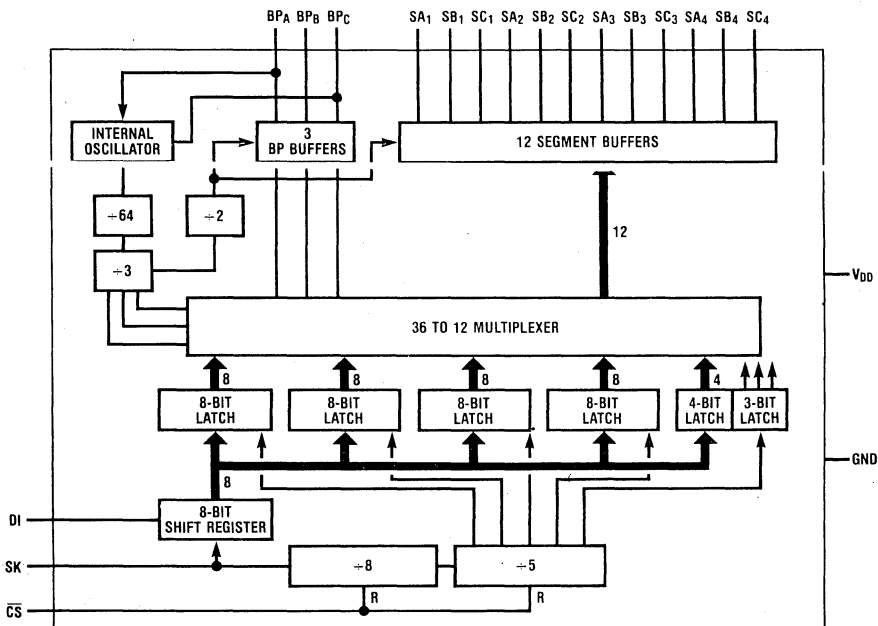
### General Description

The COP472 Liquid Crystal Display (LCD) Controller is a peripheral member of the COPSTM family, fabricated using CMOS technology. The COP472 drives a multiplexed liquid crystal display directly. Data is loaded serially and is held in internal latches. The COP472 contains an on-chip oscillator and generates all the multi-level waveforms for backplanes and segment outputs on a triplex display. One COP472 can drive 36 segments multiplexed as 3 × 12 (4½ digit display). Two COP472 devices can be used together to drive 72 segments (3 × 24) which could be an 8½ digit display.

### Features

- Direct interface to TRIPLEX LCD
- Low power dissipation (100µW typ.)
- Low cost
- Compatible with all COP400 processors
- Needs no refresh from processor
- On-chip oscillator and latches
- Expandable to longer displays
- Software compatible with COP470 V.F. Display Driver chip
- Operates from display voltage
- MICROWIRE™ compatible serial I/O
- 20-pin dual-in-line package

COPS and MICROWIRE are trademarks of National Semiconductor Corp.



COP472 Block Diagram

## Absolute Maximum Ratings

Voltage at CS, DI, SK pins	-0.3V to +9.5V
Voltage at all other Pins	-0.3V to $V_{DD} + 0.3V$
Operating Temperature Range	0°C to 70°C
Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 Seconds)	300°C

## DC Electrical Characteristics

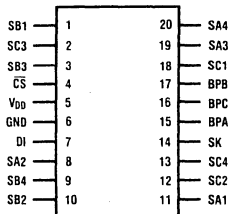
GND = 0V,  $V_{DD} = 2.4V$  to  $5.5V$ ,  $T_A = 0^\circ C$  to  $70^\circ C$   
(depends on display characteristics)

Parameter	Conditions	Min.	Max.	Units
Power Supply Voltage, $V_{DD}$		2.4	5.5	Volts
Power Supply Current, $I_{DD}$ (Note 1)	$V_{DD} = 5.5V$		250	$\mu A$
	$V_{DD} = 3V$		100	$\mu A$
Input Levels DI, SK, CS $V_{IL}$ $V_{IH}$		0.7 $V_{DD}$	0.8 9.5	Volts Volts
BPA (as Osc. In) $V_{IL}$ $V_{IH}$		$V_{DD} - 0.6$	0.6 $V_{DD}$	Volts Volts
Output Levels, BPC (as Osc. Out) $V_{OL}$ $V_{OH}$		$V_{DD} - 0.4$	0.4 $V_{DD}$	Volts Volts
Backplane Outputs (BPA, BPB, BPC) $V_{BPA, BPB, BPC}$ ON $V_{BPA, BPB, BPC}$ OFF	During BP <sup>+</sup> Time	$V_{DD} - \Delta V$ $\frac{1}{3}V_{DD} - \Delta V$	$V_{DD}$ $\frac{1}{3}V_{DD} + \Delta V$	Volts Volts
	During BP <sup>-</sup> Time	0 $\frac{2}{3}V_{DD} - \Delta V$	$\Delta V$ $\frac{2}{3}V_{DD} + \Delta V$	Volts Volts
Segment Outputs (SA <sub>1</sub> ~ SA <sub>4</sub> ) $V_{SEG}$ ON $V_{SEG}$ OFF	During BP <sup>+</sup> Time	0 $\frac{1}{3}V_{DD} - \Delta V$	$\Delta V$ $\frac{1}{3}V_{DD} + \Delta V$	Volts Volts
	During BP <sup>-</sup> Time	$V_{DD} - \Delta V$ $\frac{1}{3}V_{DD} - \Delta V$	$V_{DD}$ $\frac{1}{3}V_{DD} + \Delta V$	Volts Volts
Internal Oscillator Frequency		15	80	kHz
Frame Time (Int. Osc. + 192)		2.4	12.8	ms
Scan Frequency (1/ $T_{SCAN}$ )		39	208	Hz
SK Clock Frequency		4	250	kHz
SK Width		1.7		$\mu s$
DI Data Setup, $t_{SETUP}$ Data Hold, $t_{HOLD}$		1.0 100		$\mu s$ ns
CS $t_{SETUP}$ $t_{HOLD}$		1.0 1.0		$\mu s$ $\mu s$
Output Loading Capacitance			100	pF

**Note 1:** Power supply current is measured in stand-alone mode with all outputs open and all inputs at  $V_{DD}$ .

**Note 2:**  $\Delta V = 0.05V_{DD}$  for  $V_{DD} \geq 3V$ .  $\Delta V = 0.15V$  for  $V_{DD} < 3V$ .

**COP472  
Connection Diagram**

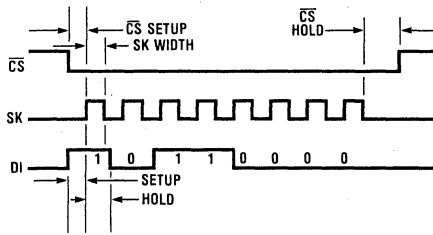


**Order Number COP472N  
NS Package N20A**

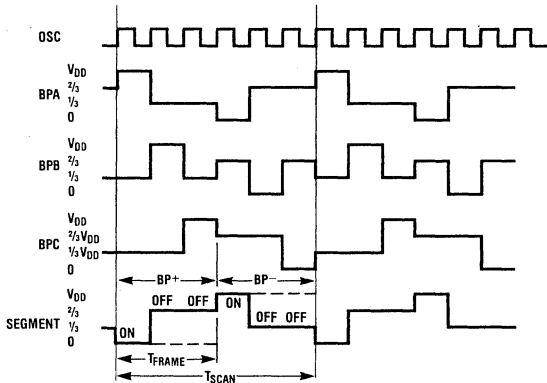
**Order Number COP472D  
NS Package D20A**

Pin	Description
$\overline{CS}$	Chip select
$V_{DD}$	Power supply (display voltage)
GND	Ground
DI	Serial data input
SK	Serial clock input
$BP_A$	Display backplane A(or oscillator in)
$BP_B$	Display backplane B
$BP_C$	Display backplane C (or oscillator out)
SA1~SC4	12 multiplexed outputs

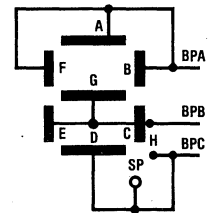
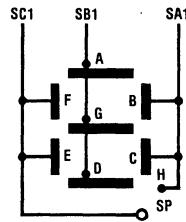
**Figure 2. Connection Diagram**



**Figure 3. Serial Load Timing Diagram**



**Figure 4. Backplane and Segment Waveforms**



**Figure 5. Typical Display Internal Connections  
Epson LD-370**

## Functional Description

The COP472 drives 36 bits of display information organized as twelve segments and three backplanes. The COP472 requires 40 information bits: 36 data and 4 control. The function of each control bit is described below. Display information format is a function of the LCD interconnections. A typical segment/backplane configuration is illustrated in Figure 5, with this configuration the COP472 will drive 4 digits of 9 segments.

To adapt the COP472 to any LCD display configuration, the segment/backplane multiplex scheme is illustrated in Table 1.

Two or more COP472 chips can be cascaded to drive additional segments. There is no limit to the number of COP472's that can be used as long as the output loading capacitance does not exceed specification.

**Table 1. COP472 Segment/Backplane Multiplex Scheme**

Bit Number	Segment, Backplane	Data to Numeric Display	
1	SA1, BPC	SH	
2	SB1, BPB	SG	
3	SC1, BPA	SF	
4	SC1, BPB	SE	
5	SB1, BPC	SD	Digit 1
6	SA1, BPB	SC	
7	SA1, BPA	SB	
8	SB1, BPA	SA	
9	SA2, BPC	SH	
10	SB2, BPB	SG	
11	SC2, BPA	SF	
12	SC2, BPB	SE	
13	SB2, BPC	SD	Digit 2
14	SA2, BPB	SC	
15	SA2, BPA	SB	
16	SB2, BPA	SA	
17	SA3, BPC	SH	
18	SB3, BPB	SG	
19	SC3, BPA	SF	
20	SC3, BPB	SE	
21	SB3, BPC	SD	Digit 3
22	SA3, BPB	SC	
23	SA3, BPA	SB	
24	SB3, BPA	SA	
25	SA4, BPC	SH	
26	SB4, BPB	SG	
27	SC4, BPA	SF	
28	SC4, BPB	SE	
29	SB4, BPC	SD	Digit 4
30	SA4, BPB	SC	
31	SA4, BPA	SB	
32	SB4, BPA	SA	
33	SC1, BPC	SP1	Digit 1
34	SC2, BPC	SP2	Digit 2
35	SC3, BPC	SP3	Digit 3
36	SC4, BPC	SP4	Digit 4
37	not used		
38	Q6		
39	Q7		
40	SYNC		

## Segment Data bits

Data is loaded in serially, in sets of eight bits. Each set of segment data is in the following format:

SA	SB	SC	SD	SE	SF	SG	SH
----	----	----	----	----	----	----	----

Data is shifted into an eight bit shift register. The first bit of the data is for segment H, digit 1. The eighth bit is segment A, digit 1. A set of eight bits is shifted in and then loaded into the digit one latches. The second set of 8 bits is loaded into digit two latches. The third set into digit three latches, and the fourth set is loaded into digit four latches.

## Control Bits

The fifth set of 8 data bits contains special segment data and control data in the following format:

SYNC	Q7	Q6	X	SP4	SP3	SP2	SP1
------	----	----	---	-----	-----	-----	-----

The first four bits shifted in contain the special character segment data. The fifth bit is not used. The sixth and seventh bits program the COP472 as a stand alone LCD driver or as a master or slave for cascading COP472's. BPC of the master is connected to BPA of each slave. The following table summarizes the function of bits six and seven:

Q7	Q6	Function	BPC Output	BPA Output
1	1	Slave	Backplane Output	Oscillator Input
0	1	Stand Alone	Backplane Output	Backplane Output
1	0	Not Used	Internal Osc. Output	Oscillator Input
0	0	Master	Internal Osc. Output	Backplane Output

The eighth bit is used to synchronize two COP472's to drive an 8½-digit display.

### Loading Sequence to Drive a 4½-Digit Display

Steps:

1. Turn  $\overline{CE}$  low.
2. Clock in 8 bits of data for digit 1.
3. Clock in 8 bits of data for digit 2.
4. Clock in 8 bits of data for digit 3.
5. Clock in 8 bits of data for digit 4.
6. Clock in 8 bits of data for special segment and control function of BPC and BPA.

0 0 1 1 SP4 SP3 SP2 SP1

7. Turn  $\overline{CS}$  high.

**Note:**  $\overline{CS}$  may be turned high after any step. For example to load only 2 digits of data, do steps 1, 2, 3, and 7.

$\overline{CS}$  must make a high to low transition before loading data in order to reset internal counters.

### Loading Sequence to Drive an 8½-Digit Display

Two or more COP472's may be connected together to drive additional segments. An eight digit multiplexed display is shown in Figure 7. The following is the loading sequence to drive an eight digit display using two COP472's. The right chip is the master and the left the slave.

Steps:

1. Turn  $\overline{CS}$  low on both COP472's.
2. Shift in 32 bits of data for for the slave's four digits.
3. Shift in 4 bits of special segment data: a zero and three ones.

1 | 1 | 1 | 0 | SP4 | SP3 | SP2 | SP1

This synchronizes both the chips and BPA is oscillator input. Both chips are now stopped.

4. Turn  $\overline{CS}$  high to both chips.
5. Turn  $\overline{CS}$  low to master COP472.
6. Shift in 32 bits of data for the master's 4 digits.
7. Shift in four bits of special segment data, a one and three zeros.

0 | 0 | 0 | 1 | SP4 | SP3 | SP2 | SP1

This sets the master COP472 to BPA as a normal backplane output and BPC as oscillator output. Now both the chips start and run off the same oscillator.

8. Turn  $\overline{CS}$  high.

The chips are now synchronized and driving 8 digits of display. To load new data simply load each chip separately in the normal manner, keeping the correct status bits to each COP472 (0110 or 0001).

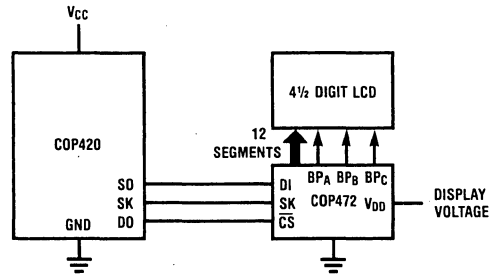


Figure 6. System Diagram — 4½ Digit Display

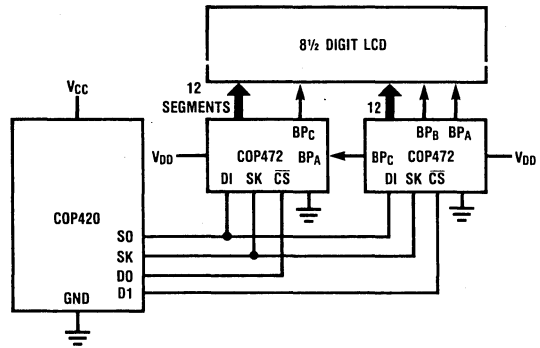


Figure 7. System Diagram — 8½ Digit Display





**Example 2**

COP420 Code to load two COP472 parts [display data is in M(0, 12)-M(0,15) and M(1, 12)-M(1, 15), special segment data is in M(0, 0) and M(1, 0)]

```

INIT:      LBI          0, 15
           OBD                          ; TURN BOTH CS'S HIGH
           LEI          8                ; ENABLE SO OUT OF S. R.
           RC
           XAS                          ; TURN OFF SK CLOCK
           LBI          3, 15           ; USE M(3, 15) FOR CONTROL BITS
           STII         7                ; STORE 7 TO SYNC BOTH CHIPS
           LBI          0, 12           ; SET B TO TURN BOTH CS'S LOW
           JSR          OUT             ; CALL OUTPUT SUBROUTINE

```

**MAIN DISPLAY SEQUENCE**

```

DISPLAY:   LBI          3, 15
           STII         8                ; SET CONTROL BITS FOR SLAVE
           LBI          0, 13           ; SET B TO TURN SLAVE CS LOW
           JSR          OUT             ; OUTPUT DATA FROM REG. 0
           LBI          3, 15
           STII         6                ; SET CONTROL BITS FOR MASTER
           LBI          1, 14           ; SET B TO TURN MASTER CS LOW
           JSR          OUT             ; OUTPUT DATA FROM REG. 1

```

**OUTPUT SUBROUTINE**

```

OUT:       OBD                          ; OUTPUT B TO CS'S
           CLRA
           AISC          12             ; 12 TO A
           CAB           ; POINT TO DISPLAY DIGIT (BD=12)

LOOP:     CLRA
           LQID          ; LOOK UP SEGMENT DATA
           CQMA          ; COPY DATA FROM Q TO M & A
           SC
           XAS          ; OUTPUT LOWER 4 BITS OF DATA
           NOP          ; DELAY
           NOP          ; DELAY
           LD           ; LOAD A WITH UPPER 4 BITS
           XAS          ; OUTPUT 4 BITS OF DATA
           NOP          ; DELAY
           NOP          ; DELAY
           RC           ; RESET C
           XAS          ; TURN OFF SK
           XIS          ; INCREMENT B FOR NEXT DISPLAY DIGIT
           JP           LOOP           ; SKIP THIS JUMP AFTER LAST DIGIT
           SC           ; SET C
           NOP
           LD           ; LOAD SPECIAL SEGS. TO A (BD=0)
           XAS          ; OUTPUT SPECIAL SEGMENTS
           NOP
           LBI          3, 15
           LD           ; LOAD A
           XAS          ; OUTPUT CONTROL BITS
           NOP
           NOP
           RC
           XAS          ; TURN OFF SK
           OBD          ; TURN CS'S HIGH (BD=15)
           RET

```



## COP498/COP398 Low Power CMOS RAM and Timer (RAT™) COP499/COP399 Low Power CMOS Memory

### General Description

The COP498/398 Low Power CMOS RAM and Timer (RAT) and the COP499/399 Memory are peripheral members of the COPS™ family, fabricated using low power CMOS technology. These devices provide external data storage and/or timing, and are accessed via the simple MICROWIRE™ serial interface. Each device contains 256 bits of read/write memory organized into 4 registers of 64 bits each; each register can be serially loaded or read by a COPS controller.

The COP498/398 also contain a crystal-based timer for timekeeping purposes, and can provide a "wake-up" signal to turn on a COPS controller. Hence, these devices are ideal for applications requiring very low power drain in a standby mode, while maintaining a real-time clock (e.g., electronically-tuned automobile radio). Power is minimized by cycling controller power off for periods of time when no processing is required.

The COP499/399 contain circuitry that enables the user to turn a controller on and off while maintaining the integrity of the memory.

A COP400 series N-channel microcontroller coupled with a COP498 (or 499) RAM/Timer offers a user the low-power advantages of an all CMOS system and the low-cost advantage of an NMOS system. This type of system is ideally suited to a wide variety of automotive and instrumentation applications.

TRI-STATE is a registered trademark of National Semiconductor Corp.  
COPS, MICROWIRE, and RAT are trademarks of National Semiconductor Corp.

### Features

- Low power dissipation
- Quiescent current = 40 nA typical (25°C,  $V_{CC} = 3.0V$ )
- Low cost
- Single supply operation (2.4V-5.5V)
- CMOS-compatible I/O
- 4 × 64 serial read/write memory
- Crystal-based selectable timer — 2.097152 MHz or 32.768 kHz (COP498/398)
- Software selectable 1 Hz or 16 Hz "wake-up" signal for COPS controller (COP498/398)
- External override to "wake-up" controller
- Compatible with all COP400 processors (processor  $V_{CC} \leq 9.5V$ )
- MICROWIRE-compatible serial I/O
- Memory protection with write enable and write disable instructions
- 14-pin dual-in-line package (COP498/398) or 8-pin dual-in-line package (COP499/399)

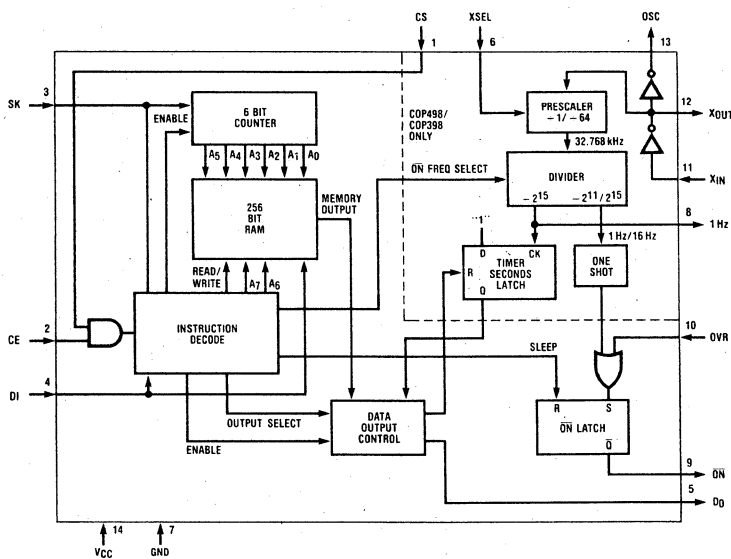


Figure 1. Block Diagram

## Absolute Maximum Ratings

Voltage relative to GND	
At XSEL, 1 Hz, X <sub>IN</sub> , X <sub>OUT</sub> , DO	-0.3V to V <sub>CC</sub> + 0.3V
At all other pins	-0.3V to 10V
Maximum V <sub>CC</sub> Voltage	6.5V
Total Sink Current Allowed	15 mA
Total Source Current Allowed	10 mA
Ambient Operating Temperature	
COP398/COP399	-40°C to +85°C
COP498/COP499	0°C to +70°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C
Power Dissipation	50 mW

"Absolute maximum ratings" indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not insured when operating the device at absolute maximum ratings.

## DC Electrical Characteristics

COP398/COP399: -40°C ≤ T<sub>A</sub> ≤ +85°C unless otherwise specified.

COP498/COP499: 0°C ≤ T<sub>A</sub> ≤ +70°C unless otherwise specified.

Parameter	Conditions	Min.	Max.	Units
Operating Voltage	COP498/COP499	2.4	5.5	V
	COP398/COP399	3.0	5.5	V
Quiescent Current  (COP398/COP399 only)	All inputs at GND			
	T <sub>A</sub> = 25°C, V <sub>CC</sub> = 3.0V		1.0	μA
	T <sub>A</sub> = 25°C, V <sub>CC</sub> = 5.0V		3.0	μA
	T <sub>A</sub> = 25°C, V <sub>CC</sub> = 5.5V		6.0	μA
	T <sub>A</sub> = 70°C, V <sub>CC</sub> = 3.0V		4.0	μA
	T <sub>A</sub> = 70°C, V <sub>CC</sub> = 5.0V		10	μA
	T <sub>A</sub> = 70°C, V <sub>CC</sub> = 5.5V		20	μA
	T <sub>A</sub> = 85°C, V <sub>CC</sub> = 3.0V		8.0	μA
	T <sub>A</sub> = 85°C, V <sub>CC</sub> = 5.0V		16	μA
	T <sub>A</sub> = 85°C, V <sub>CC</sub> = 5.5V		30	μA
COP498/COP398 Standby Current (sleep mode) (running with crystal)	V <sub>CC</sub> = Min., Osc. = 2.097 MHz		200	μA
	V <sub>CC</sub> = Max., Osc. = 2.097 MHz		700	μA
	V <sub>CC</sub> = Min., Osc. = 32.768 kHz		20	μA
	V <sub>CC</sub> = Max., Osc. = 32.768 kHz		100	μA
Operating Current	SK = 250 kHz square wave			
	V <sub>CC</sub> = Min., Osc. = 2.097 MHz		300	μA
	V <sub>CC</sub> = Max., Osc. = 2.097 MHz		920	μA
	V <sub>CC</sub> = Min., Osc. = 32.768 kHz		120	μA
V <sub>CC</sub> = Max., Osc. = 32.768 kHz		320	μA	
COP499/COP399 Operating Current	SK = 250 kHz square wave			
	V <sub>CC</sub> = Min. V <sub>CC</sub> = Max.		100 250	μA μA
Input Voltage Levels	CE Input (Schmitt Trigger Input)			
	Logic High (V <sub>IH</sub> )	0.8 V <sub>CC</sub>		V
	Logic Low (V <sub>IL</sub> )		0.4 V <sub>CC</sub>	V
	OVR Input (Schmitt Trigger Input)			
	Logic High (V <sub>IH</sub> )	0.8 V <sub>CC</sub>		V
	Logic Low (V <sub>IL</sub> )		0.2 V <sub>CC</sub>	V
	All Other Inputs			
	Logic High (V <sub>IH</sub> )	0.7 V <sub>CC</sub>		V
Logic Low (V <sub>IL</sub> )		0.3 V <sub>CC</sub>	V	

## DC Electrical Characteristics (cont'd)

Parameter	Conditions	Min.	Max.	Units
Output Voltage Levels — DO, 1 Hz CMOS Operation Logic High ( $V_{OH}$ ) Logic Low ( $V_{OL}$ )	$I_{OH} = -10 \mu A$ $I_{OL} = 10 \mu A$	$V_{CC} - 0.1$	0.1	V V
Input Leakage Current	COP498/COP499, $V_{IH} = V_{CC}$ , $V_{IL} = 0V$ COP398/COP399, $V_{IH} = V_{CC}$ , $V_{IL} = 0V$	-1.0 -2.0	+1.0 +2.0	$\mu A$ $\mu A$
TRI-STATE®, Open Drain Leakage Current	COP498/COP499, $V_H = V_{CC}$ , $V_L = 0V$ COP398/COP399, $V_H = V_{CC}$ , $V_L = 0V$	-2.5 -5.0	+2.5 +5.0	$\mu A$ $\mu A$
Output Current Levels Sink Current OSC ON $X_{OUT}$ $X_{OUT}$ 1 Hz, DO Source Current ON $X_{OUT}$ $X_{OUT}$ 1 Hz, DO	$V_{CC} = 4.5V$  $V_{OL} = 0.4V$ $V_{OL} = 1.5V$ $XSEL = 1$ , $X_{IN} = 4.5V$ , $V_{OL} = 1.0V$ $XSEL = 0$ , $X_{IN} = 4.5V$ , $V_{OL} = 2.0V$ $V_{OL} = 0.8V$  $V_{OH} = 1.0V$ $XSEL = 1$ , $X_{IN} = 0V$ , $V_{OH} = 3.0V$ $XSEL = 0$ , $X_{IN} = 0V$ , $V_{OH} = 3.0V$ $V_{OH} = 2.0V$	0.5 1.5 0.25 8.0 0.8  60 0.27 10 0.4	7.5	mA mA mA $\mu A$ mA mA $\mu A$ mA mA mA

## AC Electrical Characteristics

COP398/COP399:  $-40^\circ C \leq T_A \leq +85^\circ C$  unless otherwise specified.COP498/COP499:  $0^\circ C \leq T_A \leq +70^\circ C$  unless otherwise specified.

Parameter	Conditions	Min.	Max.	Units
COP Interface				
SK Frequency	CS = 1, CE = 1, COP498/COP499	4.096	250	kHz
	CS = 1, CE = 1, COP398/COP399	8.192	250	kHz
SK Duty Cycle	SK frequency $\geq 25$ kHz	25	75	%
	SK frequency = Min.	48	52	%
Inputs				
CS				
$t_{CSS}$		0.2		$\mu s$
$t_{CSH}$		0		$\mu s$
DI				
$t_{SETUP}$		0.4		$\mu s$
$t_{HOLD}$		0.4		$\mu s$
Output				
DO	$C_L = 100$ pF, $4.5V \leq V_{CC} \leq 5.5V$ , $V_{OH} = 0.7 V_{CC}$ , $V_{OL} = 0.4V$		2.0	$\mu s$
	$C_L = 50$ pF, $V_{CC} = \text{Min.}$ , $V_{OH} = 2V$ , $V_{OL} = 0.7V$		2.4	$\mu s$
Crystal Osc. Frequency	XSEL = 1		2.1	MHz
	XSEL = 0		65	kHz

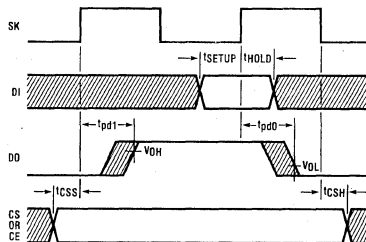
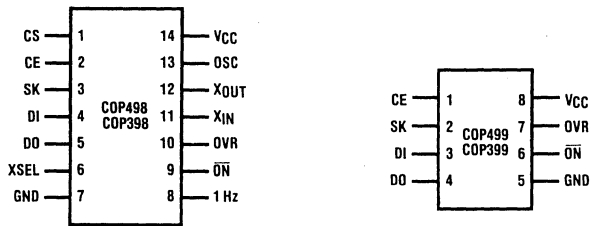


Figure 2. Synchronous Data Timing



Order Number COP498N, COP398N NS Package N14A      Order Number COP499N, COP399N NS Package N08A

Figure 3. Pin Connection Diagrams

Pin	Description	Pin	Description
CS	Chip Select	1 Hz	1 Hz Square Wave Output
CE	Chip Enable	$\overline{\text{ON}}$	Active Low Wake-Up Signal to COPS™ Controller
SK	Serial Data Clock	OVR	External Override Wake-Up for COPS Controller
DI	Serial Data Input	OSC	Open Drain Oscillator Output
DO	Serial Data Output	V <sub>CC</sub>	Power Supply
XSEL	Crystal Option Select	GND	Ground
X <sub>IN</sub>	Crystal Oscillator Input		
X <sub>OUT</sub>	Crystal Oscillator Output		

COP398 and COP399 are extended temperature devices (-40°C to +85°C) of COP498 and COP499 (0°C to 70°C) respectively, with all other functional and electrical characteristics being the same. Therefore, no further attempt will be made to distinguish between COP498 and COP398 or between COP499 and COP399. Unless otherwise specified, the following descriptions will apply to both COP498 and COP499, and they will be known as the device.

**Instruction Set**

COP498 has six instructions as indicated in Figure 4. Note that the MSB of any given instruction is a "1". This bit is properly viewed as a start bit in the interface sequence. The lower 4 bits of the instruction contain the command for the device. One of the instructions (TSEC) should not be used in COP499 as it serves no purpose.

Instruction	Opcode	Comments
	MSB	
WRITE	1 s 1 r <sub>1</sub> r <sub>0</sub>	s= $\overline{\text{ON}}$ (wake up signal) frequency select 1=16 Hz, 0=1 Hz (s selection for COP498 only) (s=0 for COP499)
READ	1 1 0 r <sub>1</sub> r <sub>0</sub>	r <sub>1</sub> , r <sub>0</sub> = register number (00, 01, 10, 11)
WREN	1 0 0 1 1	Write enable
WRDS	1 0 0 0 0	Write disable
TSEC	1 0 0 1 0	Test timer seconds latch (COP498 only)
SLEEP	1 0 0 0 1	Put COPS™ controller to sleep ( $\overline{\text{ON}}$ high)

Figure 4. Instruction Set

**Functional Description**

A block diagram of COP498 and COP499 is given in Figure 1. Positive logic is used. When a bit is set to the higher voltage it is a logic "1"; when a bit is reset to the lower voltage it is a logic "0". The COP498 can execute

six instructions: READ (from any one of 4 registers in memory); WRITE (to any one of 4 registers in memory); WREN (write enable); WRDS (write disable); TSEC (test and reset timer seconds latch); and SLEEP (drive ON signal high to turn off COPS™ controller). The COP499 can execute all the above instructions except TSEC. All communications with the device are via the serial MICROWIRE™ interface. Both CS and CE (CE only in COP499) must be high to enable the device. The device must be deselected between instructions — either CS and/or CE must go low to insure proper operation. The deselection of the device resets the counters and serial input register.

**Read/Write Memory**

The device has 256 bits of read/write memory. The memory is organized as 4 registers of 64 bits each. The data is accessed serially through the Data Input (DI) and Data Output (DO) pins. SK is the clock signal for data and instructions.

The memory address register can be conceived of as two registers: one two bits long and loaded directly from the instruction; the other six bits long and incremented by 1 with each SK pulse as long as the chip is selected. The two bit register does not change during the execution of a given instruction. The six bit register is reset to zero while the device is deselected. When counting, the six bit register wraps around from its maximum value back to zero. Thus memory locations are addressed relative to the number of SK pulses after the chip is selected.

The READ instruction will select one of the 4 registers (the register being identified in the instruction opcode as indicated in Figure 4) and output the contents of that register to the DO pin until the device is deselected. Note that data output from the device, as a result of a READ instruction, continues as long as the device is selected and clocks are provided. Reading more than 64 bits will cause rereading of some bits as the memory address register wraps around from the maximum value back to zero.

The WRITE instruction selects one of the 4 registers (the register being identified in the instruction opcode as indicated in Figure 4) and takes the data from the DI pin and stores that data into the memory register until the device is deselected. The write operation continues as long as the device is selected and clocks are provided. Thus writing more than 64 bits will cause a portion of the data to be overwritten.

### Timer (COP498 only)

With the XSEL pin tied high ( $V_{CC}$ ), the timer is a 21 stage counter which can divide a 2.097152 MHz signal down to 1 Hz. This creates the 1 Hz signal output. With XSEL tied low (ground), the timer is a 15 stage counter which divides a 32.768 kHz signal down to create the 1 Hz signal output. The rising edge of the 1 Hz signal is used internally to set the timer seconds latch. A wake-up signal is generated at the  $\overline{ON}$  output. This signal can be used to turn a COPS controller on. The wake-up rate is software selectable and may be either 1 Hz or 16 Hz. A bit in the WRITE instruction controls this wake-up rate (see Figure 4). By means of the SLEEP instruction a COPS controller may cause the  $\overline{ON}$  signal to go high thereby providing a means for the controller to safely turn itself off.

An override capability is present whereby the  $\overline{ON}$  pin may be prevented from going high. A "1" level at the OVR pin will force  $\overline{ON}$  to go low (or stay low) thereby causing the controller to turn on or remain on.  $\overline{ON}$  will remain low, and the controller on, as long as the OVR pin is high. To preserve timekeeping when using the override feature, a timer seconds latch is provided. This latch is set by the rising edge of the 1 Hz signal and is read and reset by the TSEC instruction. The timer seconds latch is primarily intended for use when the override feature is implemented. However, it does provide a convenient one second timer which is software testable over a common serial port.

### System Considerations

When the COPS processor is being turned on and off, during the power supply transition between ground and operating voltage, some pulses may occur at the output pins of the processor. By using the WRDS and WREN instructions, together with the higher "1" level of the CE pin, accidental writing into the memory may be prevented. This is done by disabling the write operation before going to sleep and enabling the write operation when the COPS processor starts execution. A WRDS instruction is automatically executed if the SLEEP instruction causes  $\overline{ON}$  to go high turning off the COPS processor. Furthermore, WREN instruction is disabled as long as  $\overline{ON}$  remains high.

The XSEL pin, which identifies the timer counter length, should be tied to either  $V_{CC}$  or ground depending on the

crystal input. For proper operation, the state of XSEL should not be changed while the device is in operation. If the oscillator and timer features are not used, the  $X_{IN}$  pin should be connected to the GND pin and XSEL tied to  $V_{CC}$ . If the override feature is not used the OVR pin should be connected to the GND pin.

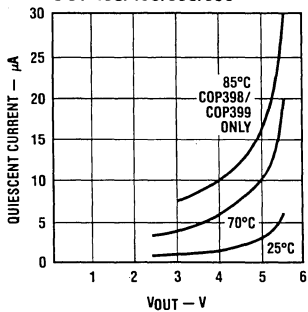
The device is in a static mode when either the CS or CE pin is low. However, the device is in a dynamic mode when both CS and CE are high and at least one high level has been detected at SK while both pins are high. Because of this, a minimum frequency is specified for the SK clock. This minimum frequency really translates to maximum on and off times for the SK clock. As the SK clock slows down, the duty cycle must get closer to 50%. For best operation, the user should regard the maximum on and off times for the SK clock as about 122  $\mu$ s (61  $\mu$ s for COP398/COP399).

### COPS™ Controller to COP498/COP499 Hardware Interface

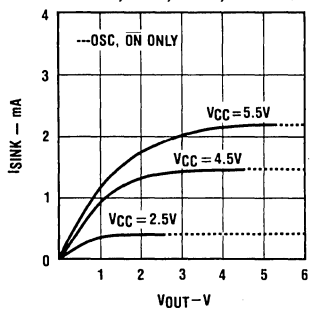
If the COPS controller is operating with a 4  $\mu$ s instruction cycle time, a 47k resistor should be connected between SK and  $V_{CC}$  to speed up the rise time of the SK clock. If the override feature is used in COP498, the override signal should be connected to the OVR pin of the COP498 and an input of the COPS controller. This is simply to provide a means for the controller to know if it was turned on by override or normal timeout. The override signal should be free of noise. In systems where the COPS controller is operating with  $V_{CC}$  greater than 6 volts, SI and the override input on the controller should have high impedance, standard TTL level input options selected. To minimize current drain in the controller, the override input to the controller should always use the high impedance option.

Figure 6a illustrates the COP498 interface in a system with supply voltage less than 6 volts. The COPS controller can either be turned on by the timer or an external signal. A PNP transistor, controlled by the  $\overline{ON}$  signal of the COP498, is used to gate the power to the COPS controller. A 0.05  $\mu$ F capacitor is connected across the supply pins of the controller to reduce voltage variations due to current spikes. It is not recommended to use large capacitance values here as problems can be introduced if the power supply fall time is too long. The switched supply fall time should be kept to about ten instruction cycles of the COPS processor. Resistor R2, between the  $\overline{ON}$  pin of the COP498 and the base of the transistor, is used to limit current. Resistor R1, between the base and emitter of the transistor, is used to turn the transistor off when  $\overline{ON}$  is high. The CE pin of the COP498 is tied to the  $V_{CC}$  pin of the controller. This guarantees that the controller is at its full operating voltage before the COP498 can be accessed. When turned on, the PNP transistor should be saturated in order to minimize the voltage drop across it. The system power supply, which here is  $V_{CC}$  to the COP498, must be high enough to insure that the controller  $V_{CC}$  — which is the system supply less the voltage drop across the PNP transistor — is high enough to be recognized as a logic "1" at the CE input of the COP498. It is also desirable to have all input signals to the COP498 as close as possible to the COP498 supply levels to eliminate any static power drain which could significantly increase standby and operating current.

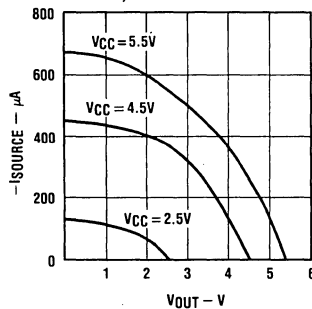
**Maximum Quiescent Current**  
COP498/499/398/399



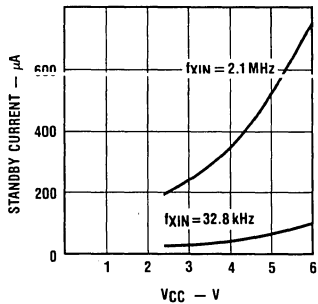
**Minimum Sink Current for DO, 1 Hz, OSC, ON**



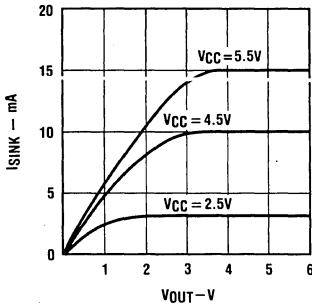
**Minimum Source Current for DO, 1 Hz**



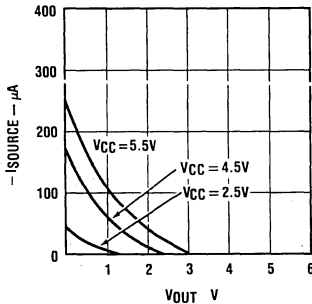
**Maximum Standby Current for COP498/398**



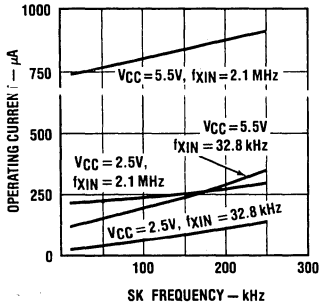
**Maximum Sink Current for ON**



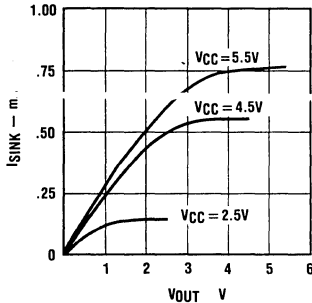
**Minimum Source Current for ON**



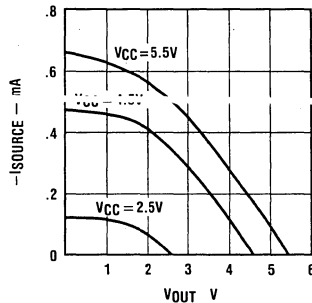
**Maximum COP498/398 Operating Current**



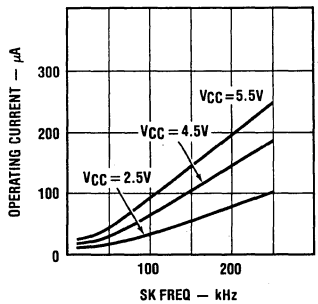
**XOUT Minimum Sink Current with XSEL = 1**



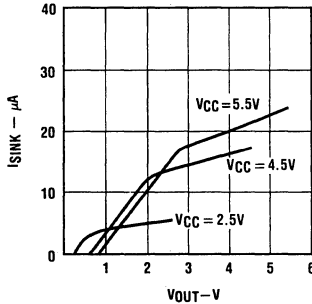
**XOUT Minimum Source Current with XSEL = 1**



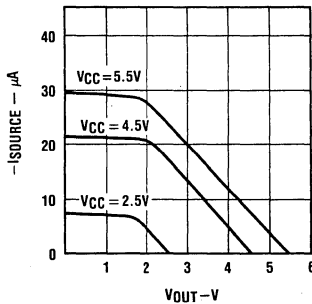
**Maximum COP499/399 Operating Current**



**XOUT Minimum Sink Current with XSEL = 0**



**XOUT Minimum Source Current with XSEL = 0**





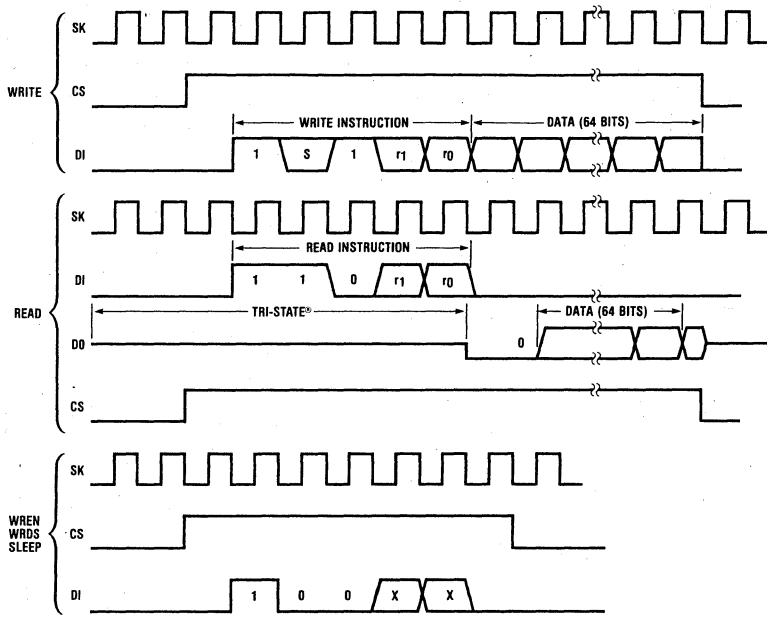


Figure 5a. Instruction Timing

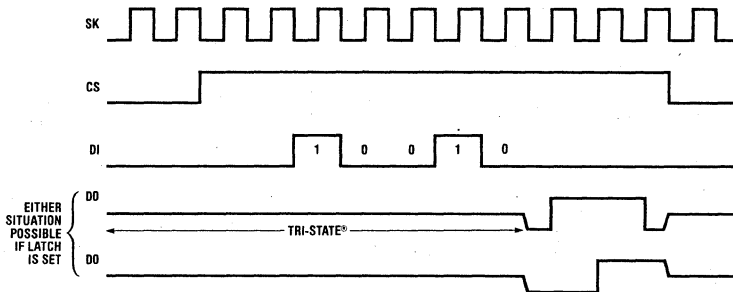


Figure 5b. TSEC Instruction Timing

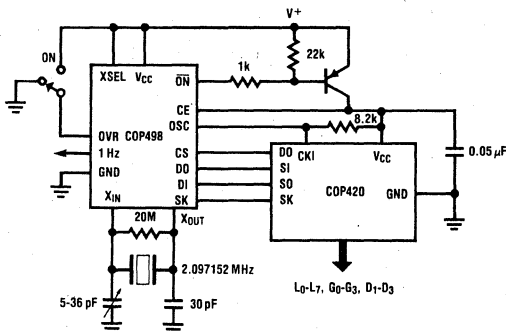


Figure 6a. COP498-COP420 Interface

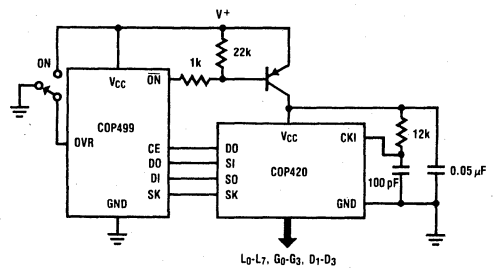


Figure 6b. COP499-COP420 Interface

TRI-STATE is a registered trademark of National Semiconductor Corp.



accomplished by means of the TSEC instruction which dumps the contents of the timer seconds latch to the serial output port and resets the latch. If the latch was set, the time must be incremented. This is accomplished by reading the appropriate memory register into the controller, incrementing the time and writing the register back out to the device. The next step is to check for the override signal. If it is present a special override routine may be performed. If no override is present, the controller turns itself off by sending a SLEEP command to the device. After sending the SLEEP command, the controller goes into a loop to wait for power to go off. In the event the controller is turned back on by the override signal before the voltage has dropped, the loop has a time limit which, when exceeded, causes the controller to jump to the beginning of the program and start again. If the override feature is not used there is no need to test the timer seconds latch nor to test for the override signal. Without the override, the controller can only be turned on by the COP498 if the time out period has elapsed. Note also that the timer features continue to operate regardless of the state of the override signal. The override signal, when high, merely forces the  $\overline{ON}$

pin to go low. The operation of the rest of the chip is in no way affected by the override signal.

### General Code for Software Interface

The code in Figure 9a is recommended for interfacing the device to any COPS controller other than COP410L/COP411L. The code in Figure 9b is the recommended interface code for COP410L/COP411L. The code is written as subroutines and the code uses one level of subroutine internally. It is apparent from the code that the software interface is somewhat different for the READ and WRITE instructions than for the rest of the instructions. The routine labelled SETUP is assumed to be in page 2 of the ROM. The rest of the code may be located anywhere in program memory subject to the usual programming rules of COPS microcontrollers. The lower four bits of the instruction opcode are assumed to be located in RAM location COMAND, which is chosen as location 3,15. Data I/O uses register 2. The controller-COP498/499 interface is assumed to be as in Figure 6 or Figure 7. It is assumed that the SIO register in the COPS controller is enabled as serial I/O prior to entry to these routines.

```

WRITE:  JSRP   SETUP
RW:     LD
        XAS           ; READ/WRITE DATA
        XIS
        JP    RW
        OBD           ; DISABLE THE COP498/499 (B = 0)
        JP    FINISH
READ:   JSRP   SETUP
        NOP           ; NEED A TOTAL OF 5 SK CLOCK DELAYS (5 NOP'S)
        NOP           ; UNTIL DATA OUT IS VALID AT SIO REGISTER
        NOP
        NOP
        NOP
        JP    RW
INSTRT: JSRP   SETUP ; ROUTINE FOR THE REST OF THE INSTRUCTIONS
        NOP
        NOP           ; DELAYS TO INSURE PROPER TIMING
FINISH: CLRA
        RC
        OBD           ; DESELECT THE COP498/499 (B = 0)
        XAS           ; TURN OFF THE CLOCK
        RET
        . PAGE 2
SETUP:  LBI    COMAND ; POINT TO LOCATION WHERE COMMAND STORED
        CLRA
        SC
        XAS           ; TURN ON SK CLOCK
        OBD           ; ENABLE THE COP498/499 (B = 15)
        CLRA
        XAS           ; MAKE SURE NO INVALID DATA SENT
        CLRA
        AISC 1       ; SET UP START BIT
        SC
        XAS           ; SEND START BIT MSD OF INSTRUCTION
        LD            ; FETCH COMMAND TO A
        NOP
        NOP           ; MAINTAIN PROPER TIMING
        XAS           ; SEND COMMAND
        LBI 2,0      ; POINT TO READ/WRITE REGISTER
        RET         ; RETURN TO MAIN ROUTINE

```

Figure 9a. Software Interface to COP498/COP499 for COPS™ Controllers Other Than COP410L/COP411L

```

WRITE: JSRP   SETUP
RW1:  XAS                      ; SEND COMMAND
RW2:  LD                      ; POSITION Bd PROPERLY
      XDS
RW:   LD
      XAS
      XIS
      JP     RW
      OBD                      ; DISABLE THE COP498/499 (B = 0)
      JP     FINISH
READ: JSRP   SETUP
      XAS                      ; SEND READ COMMAND
      NOP                      ; DELAY FOR DATA VALID
      NOP
      NOP
      NOP
      JP     RW2
INSTR:JSRP   SETUP              ; ROUTINE FOR REST OF INSTRUCTIONS
      XAS                      ; SEND INSTRUCTION
      NOP
      NOP                      ; DELAY FOR INSTRUCTION ACCEPT
      NOP
FINISH:CLRA
      RC
      OBD                      ; Deselect the COP498/499
      XAS                      ; TURN OFF THE CLOCK
      RET
      . PAGE 2
SETUP: LBI   COMAND
      CLRA
      SC
      XAS                      ; TURN ON SK CLOCK
      OBD                      ENABLE THE COP498/COP499 (B = 15)
      CLRA
      XAS                      ; MAKE SURE NO INVALID DATA SENT
      CLRA
      AISC 1
      CC
      XAS                      ; SEND START BIT-MSD OF INSTRUCTION
      LD                      ; FETCH INSTRUCTION
      LBI 2,9
      RET

```

Figure 9b. COP410L/COP411L Software Interface to COP498/COP499

### General Notes

The code in Figure 9a will read or write 64 bits at a time. Note that in the COP410L/411L the code in Figure 9b will read or write 32 bits at a time. The code of Figure 10 is recommended if the user wishes to work in blocks of 64 bits with the COP410L/411L. Only the code which is different from that shown in Figure 9b is shown in Figure 10.

The routine in Figure 10 will read/write into registers 2 and 1 in the COP410L/411L. Figure 10 illustrates the preferred method of achieving full utilization of the device memory when the COP410L/411L is the controller. Remember that all the other routines are as shown in Figure 9B. Figure 10 illustrates only that code that must be changed to achieve full usage of the device memory when using the COP410L/411L.

1. For complete safety in all cases it is recommended that the SK clock be turned off after the device has been deselected since the device is dynamic when it is enabled. If the clock is turned off while the device is selected, special care must be given to the SK timing characteristics. In no case should the clock be turned off while the device is selected if the SK period is greater than about 50  $\mu$ s.
2. The device does not become dynamic until both CS and CE are high and at least one high level is seen at the SK input. Thus the device may be safely enabled prior to turning on the clock as long as SK is low when the device is enabled.

```

WRITE: JSRP      SETUP      ; INITIALIZE, SEE FIGURE 9B
RW1:  XAS       ; SEND COMMAND
RW2:  LD        ; POSITION Bd
      XDS
RW:   LD
      XAS
      X         3          ; USE REGISTERS 2 AND 1
      LD
      NOP
      XAS
      XIS       3
      JP        RW
      OBD      ; DESELECT THE COP498/499
      JP        FINISH
    
```

Figure 10. COP410L/411L-COP498/499 Special Routine

- The device must be deselected between instructions. Failure to do so will yield improper operation. The device relies on the select lines changing state in order to clear internal registers. Only one of the select lines on the COP498 needs to go low between instructions.
- The user must insure that a WREN (write enable) instruction has been performed in order to write to the device memory. The WREN command need be given only once unless the SLEEP feature is used. If  $\overline{ON}$  goes high as a result of a SLEEP command, a write disable is automatically performed in order to provide maximum protection to the device memory while the COPSTM controller is powering up and powering down. As long as  $\overline{ON}$  remains high, WRITE and WREN instructions are disabled. Thus when the COPS controller wakes up after previously issuing a SLEEP command, a WREN instruction is required before data can be written to the device.
- The six bit section of the RAM address register will increment whenever there are clock pulses present when the CS and CE pins are high. Thus the user can position the RAM address register if he wishes by selecting the device, holding the DI pin low and supplying the appropriate number of clocks. Then, without deselecting the device, the user would send the instruction and read or write data. Although possible, this technique is not recommended as it is fairly involved.
- When using the TSEC command in COP498 with the code as given in Figure 9, the master program should test for the accumulator greater than 1 to determine if the timer seconds latch was set. Note again, test for greater than 1; do not test for greater than zero.

**Note on MICROWIRE™ Interface**

If the device is connected to a MICROWIRE interface containing other circuits whose DO (data output) pins may produce a signal swing higher than  $V_{CC}$  of the device, some protection is needed on the DO pin of the device. This happens when the DO pins of several peripherals powered by different voltages are connected together; e.g. a COP453 at 9.5V with a COP498 at 5.5V, or a COP452 at 4.5V with a COP499 at 2.4V. When the DO pin of COP498/499 is externally driven above its power supply voltage, a current will flow into it and this current must be limited to 1mA. As an example, we have two COP453's with a COP420L operating at 9.5V and a COP498 operating at 4.5V. When enabled, the DO pin of a COP453 may swing higher than 4.5V, the power supply voltage of the COP498. One way to limit the current is to use a current limiting resistor of 5.6k $\Omega$  between the DO pins of the COP453 and the COP498. NOTE: the SI pin of the COPS processor MUST BE A Hi-Z INPUT. Two configurations are possible as shown in Figure 11. Note that the resistor between DO and SI will give extra RC delay to the signal going from the DO pin to the SI pin of the COPS processor. Connection B is preferred because the DO signal from COP498 has nearly a whole SK cycle to become valid at SI input before the signal is read by the processor. When a ROMless COPS processor (COP401L/COP402/COP404L) is used for emulation, the circuit shown in Figure 12 may be used to simulate a Hi-Z input for the SI pin.

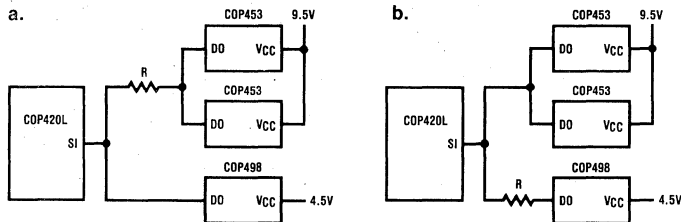


Figure 11. High Voltage Protection on DO Pin

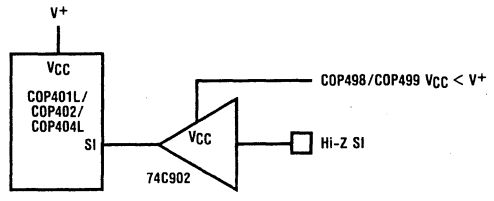


Figure 12. Simulating Hi-Z SI Input on ROMless Processors



# DS8906 AM/FM Digital Phase-Locked Loop Synthesizer

## General Description

The DS8906 is a PLL synthesizer designed specifically for use in AM/FM radios. It contains the reference oscillator, a phase comparator, a charge pump, a 120 MHz ECL/ $I^2L$  dual modulus programmable divider, and a 20-bit shift register/latch for serial data entry. The device is designed to operate with a serial data controller generating the necessary division codes for each frequency, and logic state information for radio function inputs/outputs.

The Colpitts reference oscillator for the PLL operates at 4 MHz. A chain of dividers is used to generate a 500 kHz clock signal for the external controller. Additional dividers generate a 12.5 kHz reference signal for FM and a 500 Hz reference signal for AM/SW. One of these reference signals is selected by the data from the controller for use by the phase comparator. Additional dividers are used to generate a 50 Hz timing signal used by the controller for "time-of-day".

Data is transferred between the frequency synthesizer and the controller via a 3 wire bus system. This consists of a data input line, an enable line, and a clock line. When the enable line is low, data can be shifted from the controller into the frequency synthesizer. When the enable line is transitioned from low to high, data entry is disabled and data present in the shift register is latched.

From the controller 22-bit data stream, the first 2 bits address the device permitting other devices to share the same bus. Of the remaining 20-bit data word, the next 14-bits are used for the PLL divide code. The remaining 6 bits are connected via latches to output pins. These 6 bits can be used to drive radio functions such as gain, mute, FM, AM, LW and SW only. These outputs are open collector. Bit 18 is used internally to select the AM or FM local oscillator input and to select between the 500 Hz and 12.5 kHz reference. A high level at bit 18 indicates FM and a low level indicates AM.

The PLL consists of a 14-bit programmable  $I^2L$  divider, an ECL phase comparator, an ECL dual modulus ( $p/p + 1$ ) prescaler, and a high speed charge pump. The programmable divider divides by  $(N+1)$ ,  $N$  being the number loaded into the shift register (bits 1-14 after address). It is clocked by the AM input via an ECL  $\div 7/8$  prescaler, or through a  $\div 63/64$  prescaler from the FM input. The AM input will work at frequencies up to 8 MHz, while the FM input works up to 120 MHz. The AM band is tuned with a frequency resolution of 500 Hz and the FM band is tuned with a resolution of 12.5 kHz. The buffered AM and FM inputs are self-biased and can be driven directly by the VCO thru a capacitor. The ECL phase comparator produces very accurate resolution of the phase difference between the input signal and the reference oscillator.

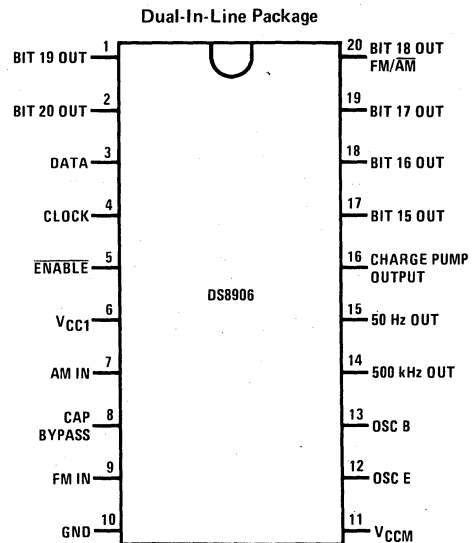
The high speed charge pump consists of a switchable constant current source ( $-0.3$  mA) and a switchable constant current sink ( $+0.3$  mA). If the VCO frequency is low, the charge pump will source current, and sink current if the VCO frequency is high.

A separate  $V_{CCM}$  pin (typically drawing 1.5 mA) powers the oscillator and reference chain to provide controller clocking frequencies when the balance of the PLL is powered down.

## Features

- Uses inexpensive 4 MHz reference crystal
- $F_{IN}$  capability greater than 120 MHz allows direct synthesis at FM frequencies
- FM resolution of 12.5 kHz allows usage of 10.7 MHz ceramic filter distribution.
- Serial data entry for simplified control.
- 50 Hz output for "time-of-day" reference with separate low power supply ( $V_{CCM}$ ).
- 6-open collector buffered outputs for band switching and other radio functions.
- Separate AM and FM inputs. AM input has 15 mV (typical) hysteresis.

## Connection Diagram



TOP VIEW

Order Number DS8906N  
NS Package N20A

## Absolute Maximum Ratings (Note 1)

Supply Voltage ( $V_{CC1}$ )	7V
( $V_{CCM}$ )	7V
Input Voltage	7V
Output Voltage	7V
Storage Temperature Range	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C

## Operating Conditions

	MIN	MAX	UNITS
Supply Voltage, $V_{CC}$			
$V_{CC1}$	4.75	5.25	V
$V_{CCM}$	4.5	6.0	V
Temperature, $T_A$	0	70	°C

## DC Electrical Characteristics (Notes 2 and 3)

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS		
$V_{IH}$	Logical "1" Input Voltage	2.1			V		
$I_{IH}$	Logical "1" Input Current	$V_{IN} = V_{CC1}$	0	10	$\mu$ A		
$V_{IL}$	Logical "0" Input Voltage			0.7	V		
$I_{IL}$	Logical "0" Input Current	Data, Clock, and ENABLE Inputs, $V_{IN} = 0V$	-5	-25	$\mu$ A		
$I_{OH}$	Logical "1" Output Current						
	All Bit Outputs, 50 Hz Output 500 kHz Output	$V_{OH} = 5.25V$ $V_{OH} = 2.4V, V_{CCM} = 4.5V$		50 -250	$\mu$ A		
$V_{OL}$	Logical "0" Output Voltage						
	All Bit Outputs 50 Hz Output, 500 kHz Output	$I_{OL} = 5\text{ mA}$ $I_{OL} = 250\ \mu\text{A}$		0.5 0.5	V		
$I_{CC1}$	Supply Current ( $V_{CC1}$ )	All Bit Outputs High	90	160	mA		
$I_{CCM}(\text{STANDBY})$	$V_{CCM}$ Supply Current	$V_{CCM} = 6.0V$ , All Other Pins Open	1.5	4.0	mA		
$I_{OUT}$	Charge Pump Output Current	$1.2V \leq V_{OUT} \leq V_{CCM} - 1.2V$ $V_{CCM} \leq 6.0V$	Pump Up	-0.10	-0.30	-0.6	mA
			Pump Down	0.10	0.30	0.6	mA
			TRI-STATE®	0		$\pm 100$	nA
$I_{CCM}(\text{OPERATE})$	$V_{CCM}$ Supply Current	$V_{CCM} = 6.0V, V_{CC1} = 5.25V$ , All Other Pins Open	2.5	6.0	mA		

TRI-STATE® is a registered trademark of National Semiconductor Corp.

AC Electrical Characteristics  $V_{CC} = 5V, T_A = 25^\circ\text{C}, t_r \leq 10\text{ ns}, t_f \leq 10\text{ ns}$ 

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	
$V_{IN}(\text{MIN})(F)$	$F_{IN}$ Minimum Signal Input	AM and FM Inputs, $0^\circ\text{C} \leq T_A \leq 70^\circ\text{C}$				
$V_{IN}(\text{MAX})(F)$	$F_{IN}$ Maximum Signal Input	AM and FM Inputs, $0^\circ\text{C} \leq T_A \leq 70^\circ\text{C}$		1000	1500	
$F_{\text{OPERATE}}$	Operating Frequency Range. (Sine Wave Input)	$V_{IN} = 100\text{ mV rms}$	AM	0.4	8	MHz
		$0^\circ\text{C} \leq T_A \leq 70^\circ\text{C}$	FM	60	120	MHz
$R_{IN}(\text{FM})$	AC Input Resistance, FM	120 MHz, $V_{IN} = 100\text{ mV rms}$	300		$\Omega$	
$R_{IN}(\text{AM})$	AC Input Resistance, AM	2 MHz, $V_{IN} = 100\text{ mV rms}$	1000		$\Omega$	
$C_{IN}$	Input Capacitance, FM and AM	$V_{IN} = 120\text{ MHz}$	3	6	10	pF
$t_{\overline{\text{EN}}1}$	Minimum ENABLE High Pulse Width		625	1250	ns	
$t_{\overline{\text{EN}}0}$	Minimum ENABLE Low Pulse Width		375	750	ns	
$t_{\text{CLK}\overline{\text{EN}}0}$	Minimum Time Before ENABLE Goes Low that CLOCK Must be Low		-50	0	ns	
$t_{\overline{\text{EN}}0\text{CLK}}$	Minimum Time After ENABLE Goes Low that CLOCK Must Remain Low		275	550	ns	
$t_{\text{CLK}\overline{\text{EN}}1}$	Minimum Time Before ENABLE Goes High that Last Positive CLOCK Edge May Occur		300	600	ns	
$t_{\overline{\text{EN}}1\text{CLK}}$	Minimum Time After ENABLE Goes High Before an Unused Positive CLOCK Edge May Occur		175	350	ns	



**AC Electrical Characteristics** (Continued)  $V_{CC} = 5V, T_A = 25^\circ C, t_r \leq 10 ns, t_f \leq 10 ns$

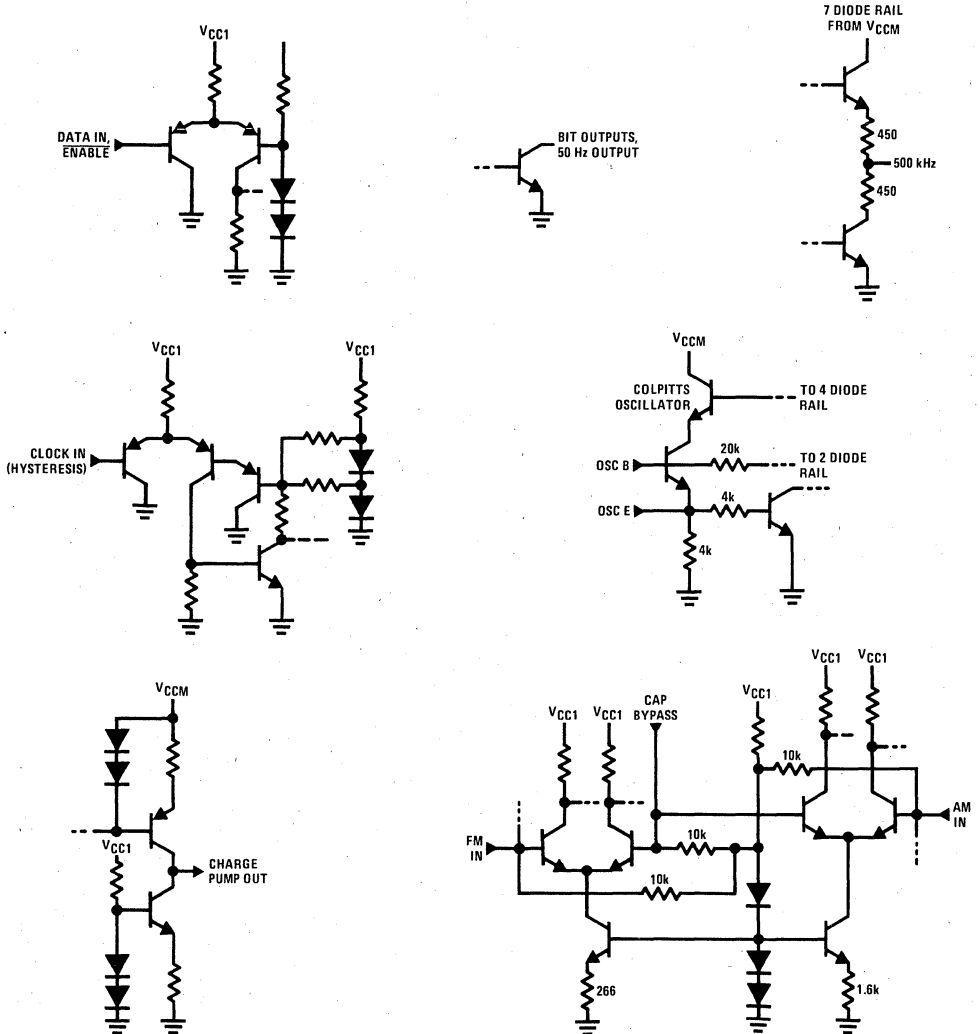
PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
tCLKH	Minimum CLOCK High Pulse Width		275	550	ns
tCLKL	Minimum CLOCK Low Pulse Width		400	800	ns
tDS	Minimum DATA Setup Time, Minimum Time Before CLOCK that DATA Must be Valid		150	300	ns
tDH	Minimum DATA Hold Time, Minimum Time After CLOCK that DATA Must Remain Valid		400	800	ns

**Note 1:** "Absolute Maximum Ratings" are those values beyond which the safety of the device cannot be guaranteed. Except for "Operating Temperature Range" they are not meant to imply that the devices should be operated at these limits. The table of "Electrical Characteristics" provides conditions for actual device operation.

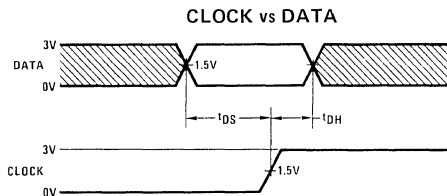
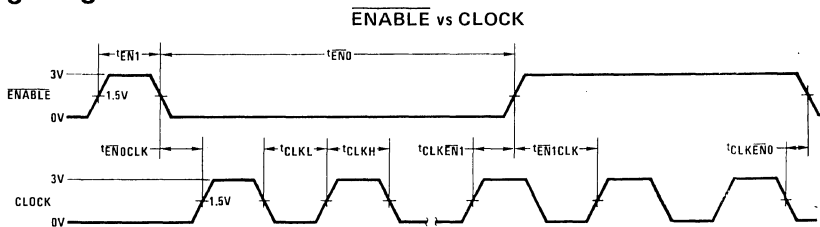
**Note 2:** Unless otherwise specified min/max limits apply across the 0°C to +70°C temperature range for the DS8906.

**Note 3:** All currents into device pins shown as positive, out of device pins as negative, all voltages referenced to ground unless otherwise noted. All values shown as max or min on absolute value basis.

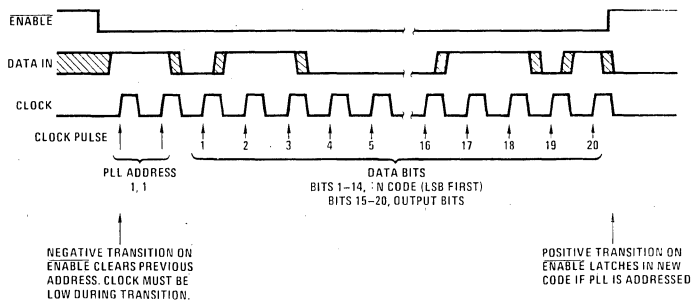
**Schematic Diagrams** (DS8906 AM/FM PLL Typical Input/Output Schematics)



Timing Diagrams \*



AM/FM Frequency Synthesizer (Scan Mode)



\* Timing diagrams are not drawn to scale. Scale within any one drawing may not be consistent, and intervals are defined positive as drawn.

SERIAL DATA ENTRY INTO THE DS8906

Serial information entry into the DS8906 is enabled by a low level on the ENABLE input. One binary bit is then accepted from the DATA input with each positive transition of the CLOCK input. The CLOCK input must be low for the specified time preceding and following the negative transition of the ENABLE input.

The first 2 bits accepted following the negative transition of the ENABLE input are interpreted as address. If these address bits are *not* 1,1, no further information will be accepted from the DATA inputs, and the internal data latches will *not* be changed when ENABLE returns high.

If these first 2 bits are 1,1, then all succeeding bits are accepted as data, and are shifted successively into the internal shift register as long as ENABLE remains low.

Any data bits preceding the 20th to last bit will be shifted out, and are thus irrelevant. Data bits are counted as any bits following 2 valid (1,1) address bits with the ENABLE low.

When the ENABLE input returns high, any further serial data input is inhibited. Upon this positive transition of the ENABLE, the data in the internal shift register is transferred into the internal data latches.

Note that until this time, the states of the internal data latches have remained unchanged.

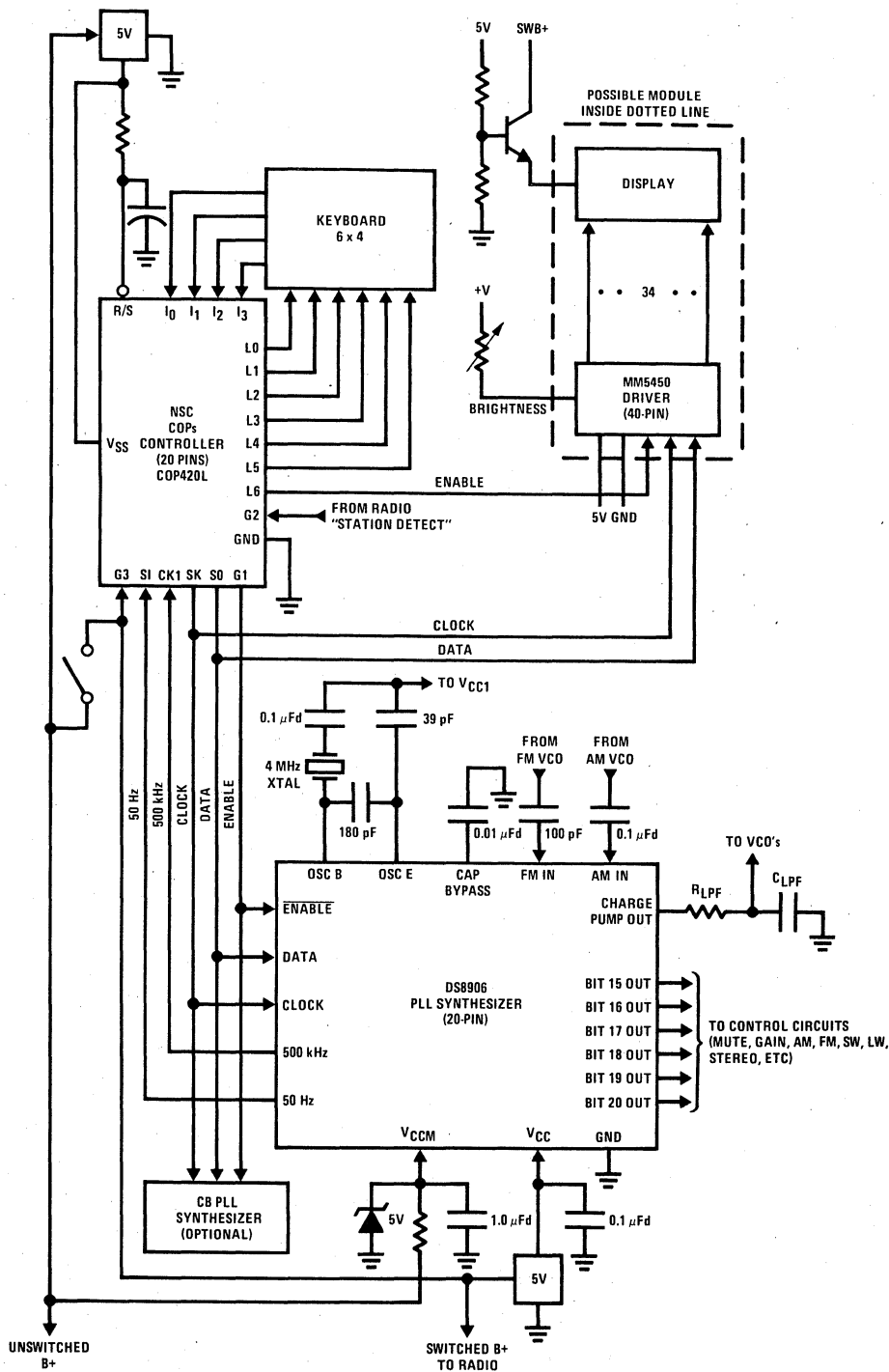
These data bits are interpreted as follows:

DATA BIT POSITION	DATA INTERPRETATION	
Last	Bit 20 Output (Pin 2)	
2nd to Last	Bit 19 Output (Pin 1)	
3rd to Last	Bit 18 Output (FM/AM) (Pin 20)	
4th to Last	Bit 17 Output (Pin 19)	
5th to Last	Bit 16 Output (Pin 18)	
6th to Last	Bit 15 Output (Pin 17)	
7th to Last	MSB of N (2 <sup>13</sup> )	
8th to Last		(2 <sup>12</sup> )
9th to Last		(2 <sup>11</sup> )
10th to Last		(2 <sup>10</sup> )
11th to Last		(2 <sup>9</sup> )
12th to Last		(2 <sup>8</sup> )
13th to Last		(2 <sup>7</sup> )
14th to Last		(2 <sup>6</sup> )
15th to Last		(2 <sup>5</sup> )
16th to Last		(2 <sup>4</sup> )
17th to Last	(2 <sup>3</sup> )	
18th to Last	(2 <sup>2</sup> )	
19th to Last	(2 <sup>1</sup> )	
20th to Last	LSB of N (2 <sup>0</sup> )	

Note. The actual divide code is N+1, i.e., the number loaded plus 1.

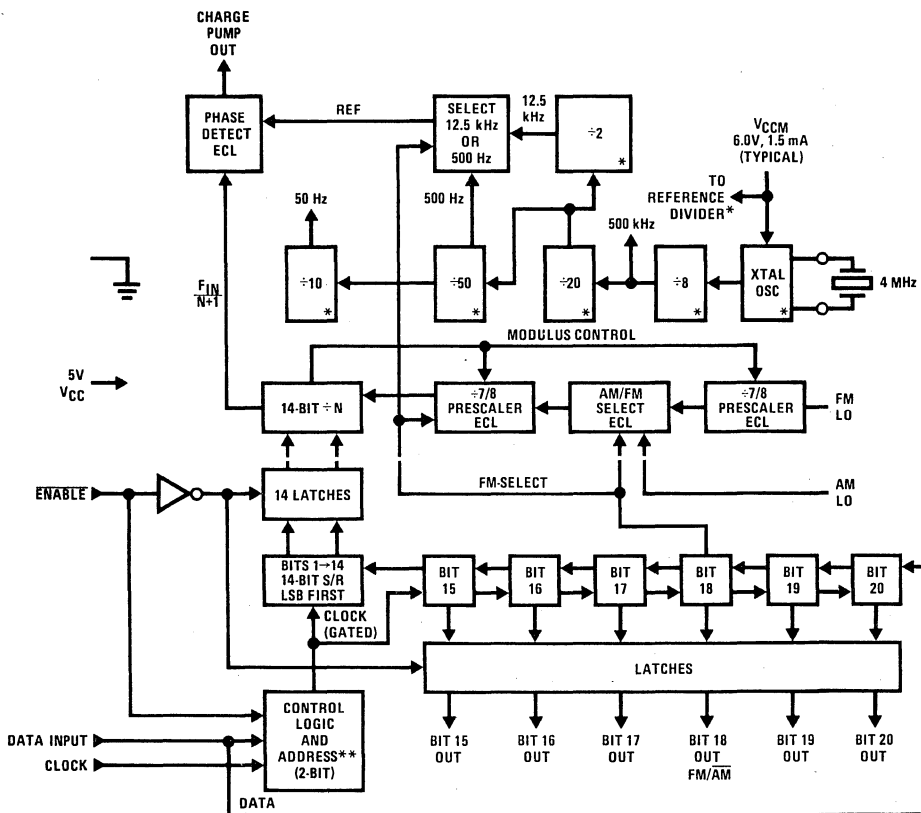
# Typical Application

Electronically Tuned Radio Controller System; Direct Drive LED



Logic Diagram

AM/FM PLL Synthesizer



\* Sections operating from V<sub>CCM</sub> supply  
 \*\* Address (1, 1)



# DS8907 AM/FM Digital Phase-Locked Loop Frequency Synthesizer

## General Description

The DS8907 is a PLL synthesizer designed specifically for use in AM/FM radios. It contains the reference oscillator, a phase comparator, a charge pump, a 120 MHz ECL/ $1^2$ L dual modulus programmable divider, and an 18-bit shift register/latch for serial data entry. The device is designed to operate with a serial data controller generating the necessary division codes for each frequency, and logic state information for radio function inputs/outputs.

The Colpitts reference oscillator for the PLL operates at 4 MHz. A chain of dividers is used to generate a 500 kHz clock signal for the external controller. Additional dividers generate a 25 kHz reference signal for FM and a 10 kHz reference signal for AM. One of these reference signals is selected by the data from the controller for use by the phase comparator.

Data is transferred between the frequency synthesizer and the controller via a 3 wire bus system. This consists of a data input line, an enable line, and a clock line. When the enable line is low, data can be shifted from the controller into the frequency synthesizer. When the enable line is transitioned from low to high, data entry is disabled and data present in the shift register is latched.

From the controller 20-bit data stream, the first 2 bits address the device permitting other devices to share the same bus. Of the remaining 18-bit data word, the next 13 bits are used for the PLL divide code. The remaining 5 bits are connected via latches to output pins. These 5 bits can be used to drive radio functions such as gain, mute, FM, AM and stereo only. These outputs are open collector. Bit 16 is used internally to select the AM or FM local oscillator input and to select between the 10 kHz and 25 kHz reference. A high level at bit 16 indicates FM and a low level indicates AM.

The PLL consists of a 13-bit programmable  $1^2$ L divider, an ECL phase comparator, an ECL dual modulus (p/p+1) prescaler, and a high speed charge pump. The programmable divider divides by (N+1), N being the number loaded into the shift register (bits 1–13 after address). It is clocked by the AM input via an ECL  $\div 7/8$  prescaler, or through a  $\div 63/64$  prescaler from the FM input. The AM input will work at frequencies up to 15 MHz, while the FM input works up to 120 MHz. The AM band is tuned with a frequency resolution of 10 kHz and the FM band is tuned with a resolution of 25 kHz. The buffered AM and FM inputs are self biased and can be driven directly by the VCO through a capacitor. The ECL phase comparator produces very accurate resolution of the phase difference between the input signal and the reference oscillator. The high speed charge pump consists of a switchable constant

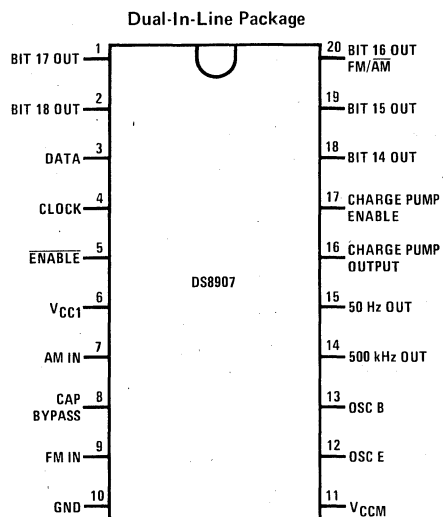
current source ( $-0.3$  mA) and a switchable constant current sink ( $+0.3$  mA). If the VCO frequency is low, the charge pump will source current, and sink current if the VCO frequency is high. When using an AFC the charge pump output may be forced into TRI-STATE<sup>®</sup> by applying a low level to the charge pump enable input.

A separate VCCM pin (typically drawing 1.5 mA) powers the oscillator and reference chain to provide controller clocking frequencies when the balance of the PLL is powered down.

## Features

- Uses inexpensive 4 MHz reference crystal
- $F_{IN}$  capability greater than 120 MHz allows direct synthesis at FM frequencies
- FM resolution of 25 kHz allows usage of 10.7 MHz ceramic filter distribution
- Serial data entry for simplified control
- 50 Hz output for "time-of-day" reference driven from separate low power VCCM
- 5-open collector buffered outputs for controlling various radio functions
- Separate AM and FM inputs. AM input has 15 mV (typical) hysteresis

## Connection Diagram



TOP VIEW

Order Number DS8907N  
NS Package N20A

**Absolute Maximum Ratings** (Note 1)

Supply Voltage (V <sub>CC1</sub> ) (V <sub>CCM</sub> )	7V
Input Voltage	7V
Output Voltage	7V
Storage Temperature Range	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C

**Operating Conditions**

	MIN	MAX	UNITS
Supply Voltage, V <sub>CC</sub>			
V <sub>CC1</sub>	4.75	5.25	V
V <sub>CCM</sub>	4.5	6.0	V
Temperature, T <sub>A</sub>	0	70	°C

**DC Electrical Characteristics** (Notes 2 and 3)

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	
V <sub>IH</sub>	Logical "1" Input Voltage	2.1			V	
I <sub>IH</sub>	Logical "1" Input Current	V <sub>IN</sub> = 2.7V	0	10	μA	
V <sub>IL</sub>	Logical "0" Input Voltage			0.7	V	
I <sub>IL</sub>	Logical "0" Input Current	Data, Clock, and $\overline{\text{ENABLE}}$ Inputs, V <sub>IN</sub> = 0V	-5	-25	μA	
I <sub>IL</sub>	Logical "0" Input Current	Charge Pump Enable, V <sub>IN</sub> = 0V	-250	-450	μA	
I <sub>OH</sub>	Logical "1" Output Current					
	All Bit Outputs, 50 Hz Output	V <sub>OH</sub> = 5.25V		50	μA	
	500 kHz Output	V <sub>OH</sub> = 2.4V, V <sub>CCM</sub> = 4.5V		-250	μA	
V <sub>OL</sub>	Logical "0" Output Voltage					
	All Bit Outputs	I <sub>OL</sub> = 5 mA		0.5	V	
	50 Hz Output, 500 kHz Output	I <sub>OL</sub> = 250 μA		0.5	V	
I <sub>CC1</sub>	Supply Current (V <sub>CC1</sub> )	All Bit Outputs High	90	160	mA	
I <sub>CCM</sub> (STANDBY)	V <sub>CCM</sub> Supply Current	V <sub>CCM</sub> = 6.0V, All Other Pins Open	1.5	4.0	mA	
I <sub>OUT</sub>	Charge Pump Output Current	1.2V ≤ V <sub>OUT</sub> ≤ V <sub>CCM</sub> - 1.2V V <sub>CCM</sub> ≤ 6.0V				
		Pump Up	-0.10	-0.30	-0.6	mA
		Pump Down	0.10	0.30	0.6	mA
		TRI-STATE <sup>®</sup>	0	±100	nA	
I <sub>CCM</sub> (OPERATE)	V <sub>CCM</sub> Supply Current	V <sub>CCM</sub> = 6.0V, V <sub>CC1</sub> = 5.25V, All Other Pins Open	2.5	6.0	mA	

**AC Electrical Characteristics** V<sub>CC</sub> = 5V, T<sub>A</sub> = 25°C, t<sub>r</sub> ≤ 10 ns, t<sub>f</sub> ≤ 10 ns

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	
V <sub>IN</sub> (MIN)(F)	F <sub>IN</sub> Minimum Signal Input	AM and FM Inputs, 0°C ≤ T <sub>A</sub> ≤ 70°C		20	100	mV (rms)
V <sub>IN</sub> (MAX)(F)	F <sub>IN</sub> Maximum Signal Input	AM and FM Inputs, 0°C ≤ T <sub>A</sub> ≤ 70°C	1000	1500		mV (rms)
F <sub>OPERATE</sub>	Operating Frequency Range (Sine Wave Input)	V <sub>IN</sub> = 100 mV rms 0°C ≤ T <sub>A</sub> ≤ 70°C				
		AM	0.4		8	MHz
		FM	60		120	MHz
R <sub>IN</sub> (FM)	AC Input Resistance, FM	120 MHz, V <sub>IN</sub> = 100 mV rms	300			Ω
R <sub>IN</sub> (AM)	AC Input Resistance, AM	2 MHz, V <sub>IN</sub> = 100 mV rms	1000			Ω
C <sub>IN</sub>	Input Capacitance, FM and AM	V <sub>IN</sub> = 120 MHz	3	6	10	pF
t <sub>EN1</sub>	Minimum $\overline{\text{ENABLE}}$ High Pulse Width			625	1250	ns
t <sub>EN0</sub>	Minimum $\overline{\text{ENABLE}}$ Low Pulse Width			375	750	ns
t <sub>CLK<math>\overline{\text{EN}}</math>0</sub>	Minimum Time Before $\overline{\text{ENABLE}}$ Goes Low that $\overline{\text{CLOCK}}$ Must be Low			-50	0	ns
t <sub>EN0CLK</sub>	Minimum Time After $\overline{\text{ENABLE}}$ Goes Low that $\overline{\text{CLOCK}}$ Must Remain Low			275	550	ns
t <sub>CLK<math>\overline{\text{EN}}</math>1</sub>	Minimum Time Before $\overline{\text{ENABLE}}$ Goes High that Last Positive $\overline{\text{CLOCK}}$ Edge May Occur			300	600	ns

# AC Electrical Characteristics (Continued) $V_{CC} = 5V, T_A = 25^\circ C, t_r \leq 10 \text{ ns}, t_f \leq 10 \text{ ns}$

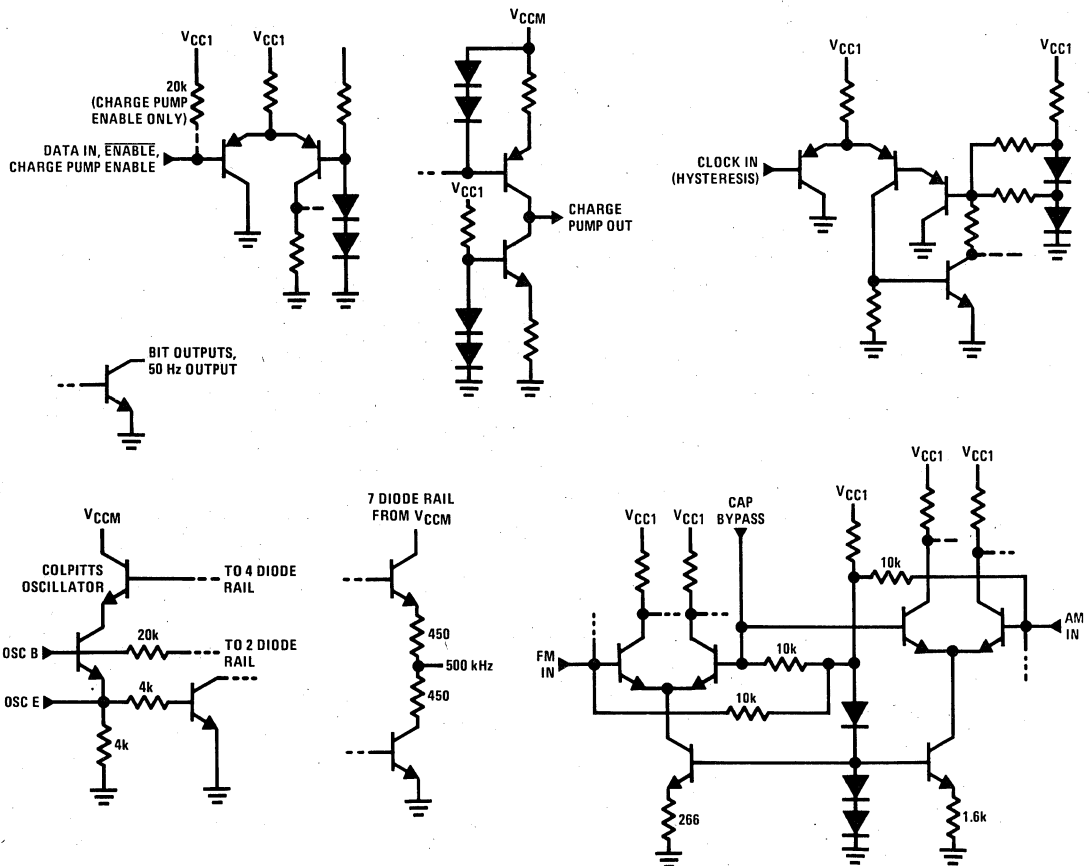
PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
tEN1CLK	Minimum Time After ENABLE Goes High Before an Unused Positive CLOCK Edge May Occur		175	350	ns
tCLKH	Minimum CLOCK High Pulse Width		275	550	ns
tCLKL	Minimum CLOCK Low Pulse Width		400	800	ns
tDS	Minimum DATA Setup Time, Minimum Time Before CLOCK that DATA Must be Valid		150	300	ns
tDH	Minimum DATA Hold Time, Minimum Time After CLOCK that DATA Must Remain Valid		400	800	ns

**Note 1:** "Absolute Maximum Ratings" are those values beyond which the safety of the device cannot be guaranteed. Except for "Operating Temperature Range" they are not meant to imply that the devices should be operated at these limits. The table of "Electrical Characteristics" provides conditions for actual device operation.

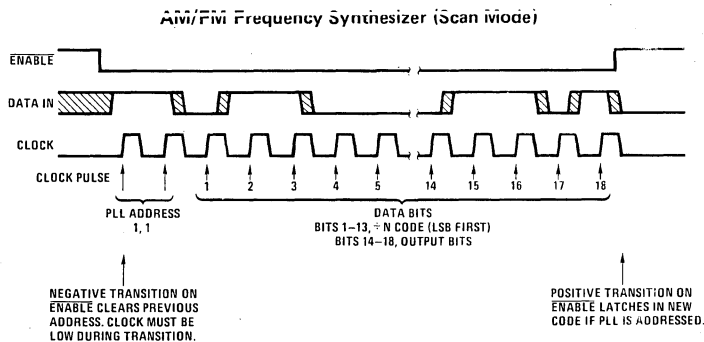
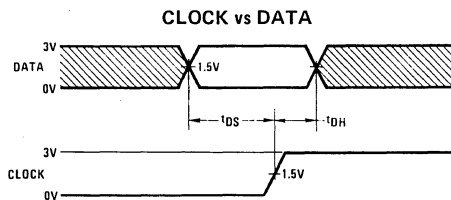
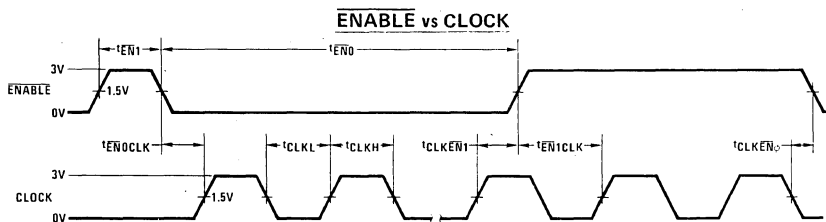
**Note 2:** Unless otherwise specified min/max limits apply across the  $-40^\circ C$  to  $+85^\circ C$  temperature range for the DS8907.

**Note 3:** All currents into device pins shown as positive, out of device pins as negative, all voltages referenced to ground unless otherwise noted. All values shown as max or min on absolute value basis.

## Schematic Diagrams (DS8907 AM/FM PLL Typical Input/Output Schematics)



## Timing Diagrams\*



\*Timing diagrams are not drawn to scale. Scale within any one drawing may not be consistent, and intervals are defined positive as drawn.

**SERIAL DATA ENTRY INTO THE DS8907**

Serial information entry into the DS8907 is enabled by a low level on the **ENABLE** input. One binary bit is then accepted from the **DATA** input with each positive transition of the **CLOCK** input. The **CLOCK** input must be low for the specified time preceding and following the negative transition of the **ENABLE** input.

The first two bits accepted following the negative transition of the **ENABLE** input are interpreted as address. If these address bits are *not* 1,1, *no* further information will be accepted from the **DATA** inputs, and the internal data latches *will not* be changed when **ENABLE** returns high.

If these first two bits are 1,1, then all succeeding bits are *accepted* as data, and are shifted successively into the internal shift register as long as **ENABLE** remains low.

Any data bits preceding the 18th to last bit will be shifted out, and thus are irrelevant. Data bits are counted as any bits *following* two valid address bits (1,1) with the **ENABLE** low. When the **ENABLE** input returns high, any further serial data entry is inhibited. Upon this positive transition, the data in

the internal shift register is transferred into the internal data latches. Note that until this time, the states of the internal data latches have remained unchanged.

These data bits are interpreted as follows:

DATA BIT POSITION	DATA INTERPRETATION
Last	Bit 18 Output (Pin 2)
2nd to Last	Bit 17 Output (Pin 1)
3rd to Last	Bit 16 Output (FM/AM) (Pin 20)
4th to Last	Bit 15 Output (Pin 19)
5th to Last	Bit 14 Output (Pin 18)
6th to Last	MSB of $\div N$ ( $2^{12}$ )
7th to Last	( $2^{11}$ )
8th to Last	( $2^{10}$ )
9th to Last	( $2^9$ )
10th to Last	( $2^8$ )
11th to Last	( $2^7$ )
12th to Last	( $2^6$ )
13th to Last	( $2^5$ )
14th to Last	( $2^4$ )
15th to Last	( $2^3$ )
16th to Last	( $2^2$ )
17th to Last	( $2^1$ )
18th to Last	LSB of $\div N$ ( $2^0$ )

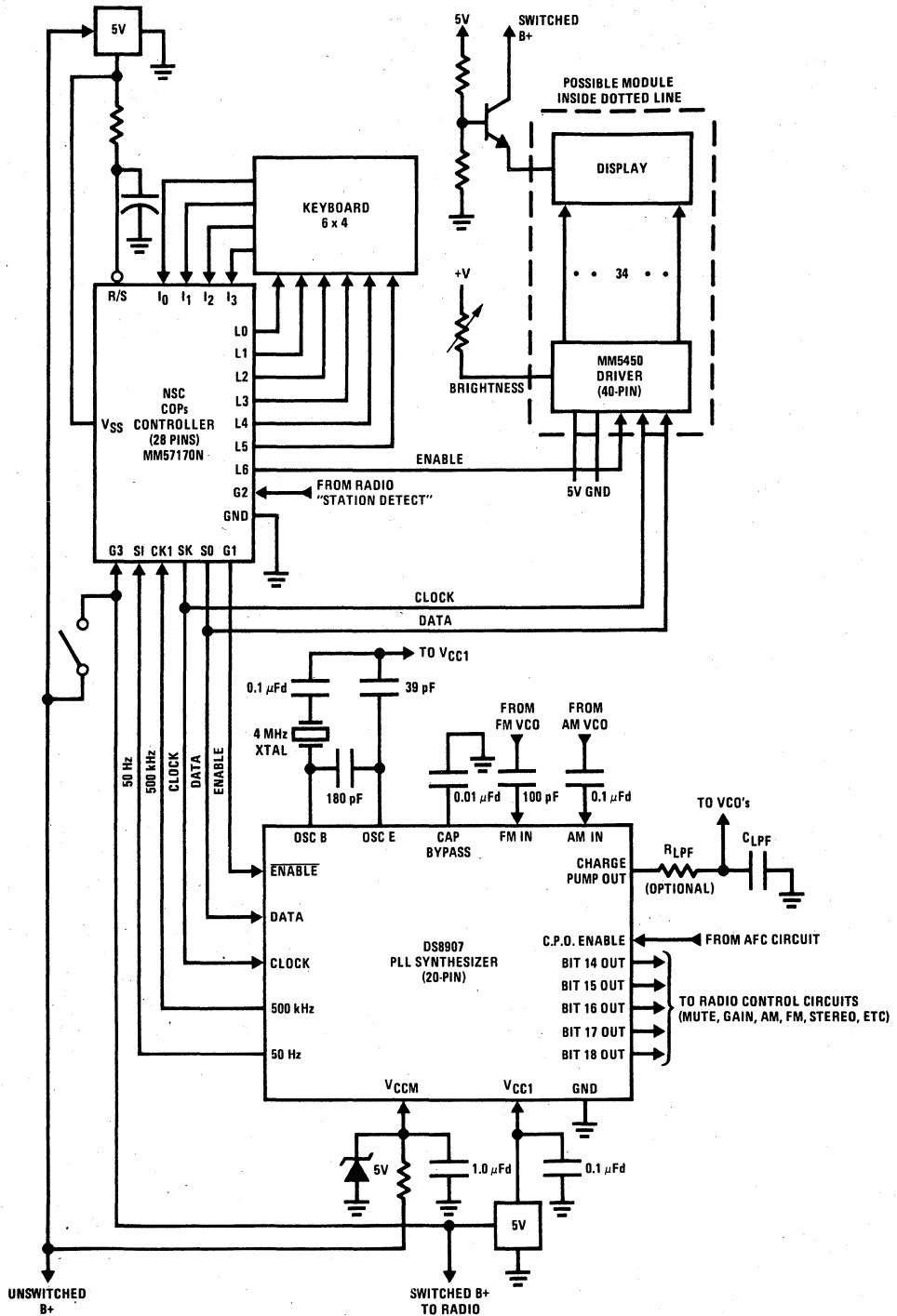
}  $\div N$

Note. The actual divide code is  $N+1$ , i.e., the number loaded plus 1.



# Typical Application

Electronically Tuned Radio Controller System; Direct Drive LED





## DS8908 AM/FM Digital Phase-Locked Loop Frequency Synthesizer

### General Description

The DS8908 is a PLL synthesizer designed specifically for use in AM/FM radios. It contains the reference oscillator, a phase comparator, a charge pump, an operational amplifier, a 120 MHz ECL/I<sup>2</sup>L dual modulus programmable divider, and a 19-bit shift register/latch for serial data entry. The device is designed to operate with a serial data controller generating the necessary division codes for each frequency, and logic state information for radio function inputs/outputs.

A 3.96 MHz pierce oscillator and divider chain generate a 1.98 MHz external controller clock, a 20 kHz, 10 kHz, 9 kHz, and 1 kHz reference signals, and a 50 Hz time-of-day signal. The oscillator and divider chain are sourced by the V<sub>CCM</sub> pin thus providing a low power controller clock drive and time-of-day indication when the balance of the PLL is powered down.

The 21-bit serial data stream is transferred between the frequency synthesizer and the controller via a 3-wire bus system comprised of a data line, a clock line, and an enable line.

The first 2 bits in the serial data stream address the synthesizer thus permitting other devices such as display drivers to share the same bus. The next 14 bits are used for the PLL (N + 1) divide code. The 15th bit is used internally to select the AM or FM local oscillator input. A high level on this bit enables the FM input and a low level enables the AM input. The 16th and 17th bits are used to select one of the 4 reference frequencies. The 18th and 19th bits are connected via latches to open collector outputs. These outputs can be used to drive radio functions such as gain, mute, AM, FM, or charge pump current source levels.

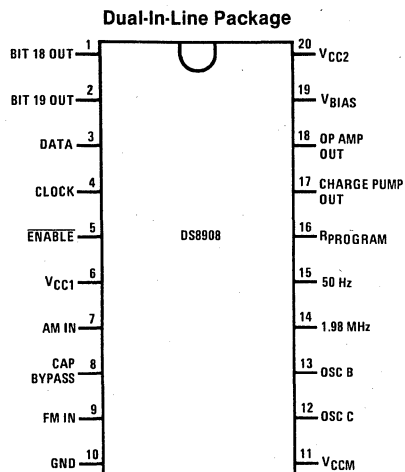
The PLL consists of a 14-bit programmable I<sup>2</sup>L divider, an ECL phase comparator, an ECL dual modulus (p/p + 1) prescaler, a high speed charge pump, and an operational amplifier. The programmable divider divides by (N + 1), N being the number loaded into the shift register. The programmable divider is clocked through a ÷7/8 prescaler by the AM input or through a ÷63/64 prescaler by the FM input. The AM input will work at frequencies up to 15 MHz, while the FM input works up to 120 MHz. The VCO can be tuned with a frequency resolution of either 1 kHz, 9 kHz, 10 kHz, or 20 kHz. The buffered AM and FM inputs are self-biased and can be driven directly by the VCO through a capacitor. The ECL phase comparator produces very accurate resolution of the phase difference between the input signal and the reference oscillator. The high speed charge pump consists of a switchable constant current source and sink. The charge pump can be programmed to deliver from 0.1 mA to 1 mA of constant current by connection of an external resistor from pin R<sub>PROGRAM</sub> to ground or any of the open collector bit outputs. Connection of programming resistors to the bit outputs enables the controller to adjust

the loop gain for the particular reference frequency selected. The charge pump will source current if the VCO frequency is high and sink current if the VCO frequency is low. The low noise operational amplifier provided has a high impedance JFET input and a large output voltage range. The op amp's negative input is common with the charge pump output and its positive input is internally biased.

### Features

- Uses inexpensive 3.96 MHz reference crystal
- F<sub>IN</sub> capability greater than 120 MHz allows direct synthesis at FM frequencies
- FM resolution of either 10 kHz or 20 kHz allows usage of 10.7 MHz ceramic filter distribution
- Serial data entry for simplified control
- 50 Hz output for time-of-day reference driven from separate low power V<sub>CCM</sub>
- 2 open collector buffered outputs for controlling various radio functions
- Separate AM and FM inputs; AM input has 15 mV (typical) hysteresis
- Programmable charge pump current sources enable adjustment of system loop gain
- Operational amplifier provides high impedance load to charge pump output and enables a high voltage output to VCO

### Connection Diagram



TOP VIEW

Order Number DS8908N  
NS Package N20A

## Absolute Maximum Ratings (Note 1)

Supply Voltage ( $V_{CC1}$ )( $V_{CCM}$ ) ( $V_{CC2}$ )	7V 17V
Input Voltage	7V
Output Voltage	7V
Storage Temperature Range	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C

## Operating Conditions

	Min	Max	Units
$V_{CC1}$	4.75	5.25	V
$V_{CC2}$	$V_{CC1} + 1.5$	15.0	V
$V_{CCM}$	3.5	5.5	V
Temperature, $T_A$	-40	+85	°C

## DC Electrical Characteristics (Notes 2 and 3)

Parameter	Conditions	Min	Typ	Max	Units	
$V_{IH}$ Logical "1" Input Voltage		2.0			V	
$I_{IH}$ Logical "1" Input Current	$V_{IN} = 2.7V$		0	10	$\mu A$	
$V_{IL}$ Logical "0" Input Voltage				0.8	V	
$I_{IL}$ Logical "0" Input Current	Data, Clock, and $\overline{ENABLE}$ Inputs, $V_{IN} = 0V$		-5	-25	$\mu A$	
$I_{OH}$ Logical "1" Output Current All Bit Outputs, 50 Hz Output 1.98 MHz Output	$V_{OH} = 5.25V$			50	$\mu A$	
	$V_{OH} = 2.4V$ , $V_{CCM} = 4.5V$			-250	$\mu A$	
$V_{OL}$ Logical "0" Output Voltage All Bit Outputs 50 Hz Output, 1.98 MHz Output	$I_{OL} = 5 mA$			0.5	V	
	$I_{OL} = 250 \mu A$			0.5	V	
$I_{CC1}$ Supply Current ( $V_{CC1}$ )	All Bit Outputs High			160	mA	
$I_{CCM}$ $V_{CCM}$ Supply Current	$V_{CCM} = 5.5V$ , All Other Pins Open		2.5	4.5	mA	
$I_{OUT}$ Charge Pump Output Current	$2.5 k\Omega \leq R_{PROG} \leq 25 k\Omega$ $V_{CC1} \leq 5.25V$ , $V_{OUT} = V_{BIAS}$ $I_{PROG} = V_{CC1}/2 R_{PROG}$	Pump Up	-20	$I_{PROG}$	+20	%
		Pump Down	-20	$I_{PROG}$	+20	%
		TRI-STATE <sup>®</sup>		0	$\pm 100$	nA
$I_{CC2}$ $V_{CC2}$ Supply Current	$V_{CCM} = 5V$ , $V_{CC1} = 5.25V$ , $V_{CC2} = 15V$ All Other Pins Open		6.7	10	mA	
$V_{OHOP}$ Maximum Output Voltage Op Amp Output	$V_{CC1} = 4.75V$ , CPO = Pump Down State	$V_{CC2} - 0.4$			V	
$V_{OLOP}$ Minimum Output Voltage Op Amp Output	$V_{CC1} = 5.25V$ , CPO = Pump Up State			0.6	V	
+ $\Delta V$ CPO Voltage Delta vs Op Amp + Current Delta	CPO Shorted to Op Amp Output $R_{PROGRAM} = 2.5 k\Omega$ CPO State: TRI-STATE vs Pump Up			40	mV	
- $\Delta V$ CPO Voltage Delta vs Op Amp-Current Delta	CPO Shorted to Op Amp Output $R_{PROGRAM} = 2.5 k\Omega$ CPO State: Pump Down vs TRI-STATE			-40	mV	

AC Electrical Characteristics  $V_{CC} = 5V$ ,  $T_A = 25^\circ C$ ,  $t_r \leq 10 ns$ ,  $t_f \leq 10 ns$ 

Parameter	Conditions	Min	Typ	Max	Units
$V_{IN(MIN)(F)}$ $F_{IN}$ Minimum Signal Input	AM and FM Inputs, $-40^\circ C \leq T_A \leq 85^\circ C$		20	100	mV(rms)
$V_{IN(MAX)(F)}$ $F_{IN}$ Maximum Signal Input	AM and FM Inputs, $-40^\circ C \leq T_A \leq 85^\circ C$	1000	1500		mV(rms)
$F_{OPERATE}$ Operating Frequency Range (Sine Wave Input)	$V_{IN} = 100 mV rms$ $-40^\circ C \leq T_A \leq 85^\circ C$	AM	0.5	8	MHz
		FM	80	120	MHz
$R_{IN}(FM)$ AC Input Resistance, FM	120 MHz, $V_{IN} = 100 mV rms$	600			$\Omega$
$R_{IN}(AM)$ AC Input Resistance, AM	15 MHz, $V_{IN} = 100 mV rms$	1000			$\Omega$
$C_{IN}$ Input Capacitance, FM and AM	$V_{IN} = 120 MHz$	3	6	10	pF
$t_{EN1}$ Minimum ENABLE High Pulse Width			625	1250	ns

TRI-STATE<sup>®</sup> is a registered trademark of National Semiconductor Corp.

# AC Electrical Characteristics (Continued) $V_{CC} = 5V, T_A = 25^\circ C, t_r \leq 10 ns, t_f \leq 10 ns$

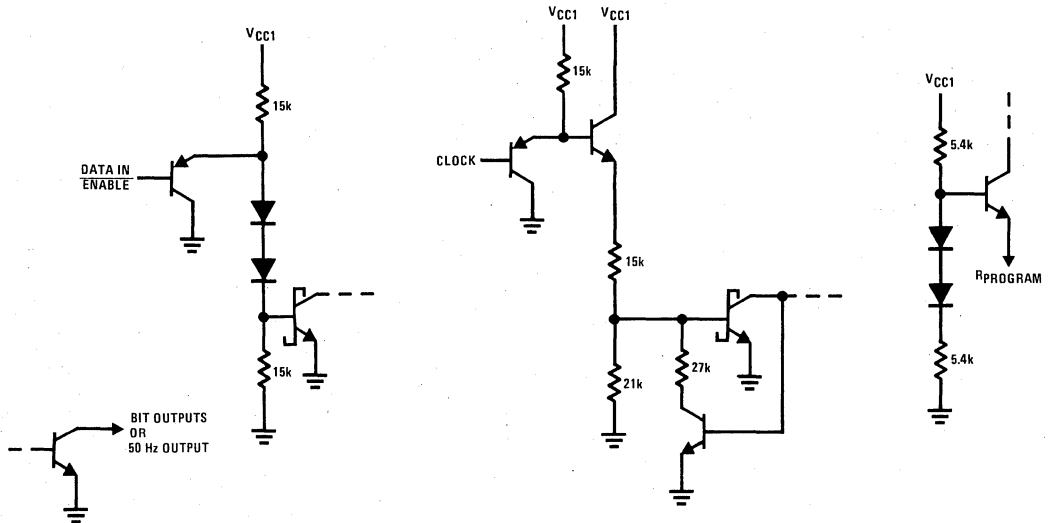
Parameter	Conditions	Min	Typ	Max	Units
$t_{EN0}$	Minimum $\overline{ENABLE}$ Low Pulse Width		375	750	ns
$t_{CLK\overline{EN}0}$	Minimum Time Before $\overline{ENABLE}$ Goes Low that CLOCK Must be Low		-50	0	ns
$t_{\overline{EN}0CLK}$	Minimum Time After $\overline{ENABLE}$ Goes Low that CLOCK Must Remain Low		275	550	ns
$t_{CLK\overline{EN}1}$	Minimum Time Before $\overline{ENABLE}$ Goes High that Last Positive CLOCK Edge May Occur		300	600	ns
$t_{\overline{EN}1CLK}$	Minimum Time After $\overline{ENABLE}$ Goes High Before an Unused Positive CLOCK Edge May Occur		175	350	ns
$t_{CLKH}$	Minimum CLOCK High Pulse Width		275	550	ns
$t_{CLKL}$	Minimum CLOCK Low Pulse Width		400	800	ns
$t_{DS}$	Minimum DATA Set-Up Time, Minimum Time Before CLOCK that DATA Must be Valid		150	300	ns
$t_{DH}$	Minimum DATA Hold Time, Minimum Time After CLOCK that DATA Must Remain Valid		400	800	ns

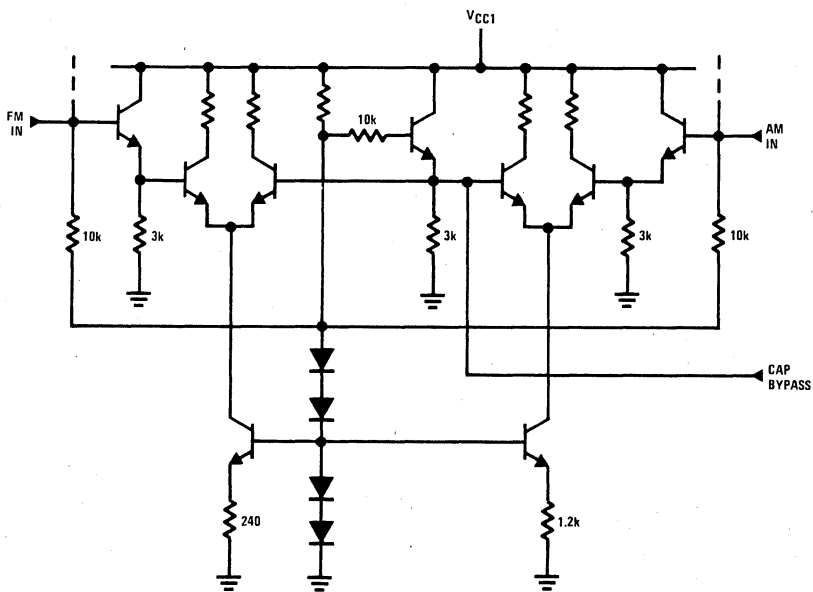
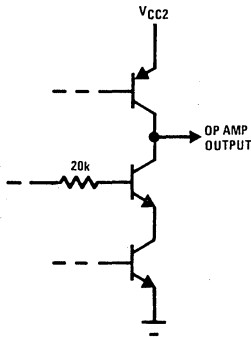
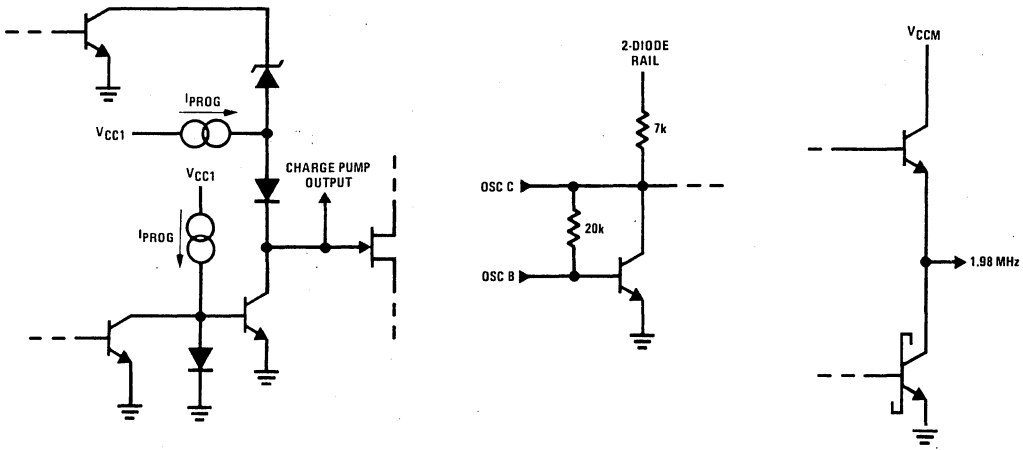
**Note 1:** "Absolute Maximum Ratings" are those values beyond which the safety of the device cannot be guaranteed. Except for "Operating Temperature Range" they are not meant to imply that the devices should be operated at these limits. The table of "Electrical Characteristics" provides conditions for actual device operation.

**Note 2:** Unless otherwise specified min/max limits apply across the  $-40^\circ C$  to  $+85^\circ C$  temperature range for the DS8908.

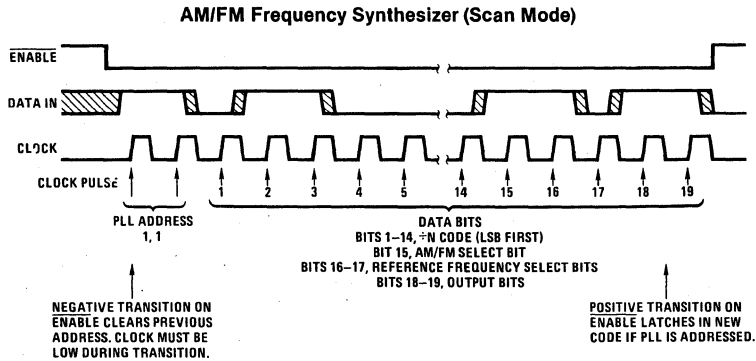
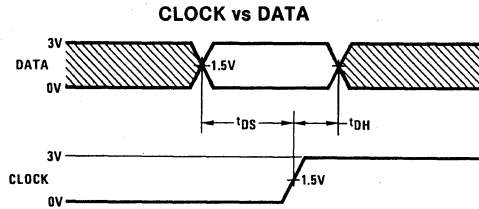
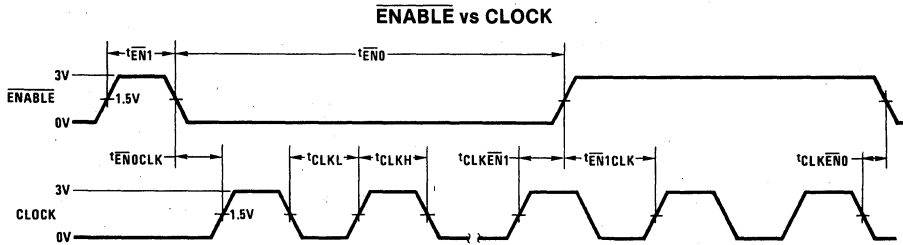
**Note 3:** All currents into device pins shown as positive, out of device pins as negative, all voltages referenced to ground unless otherwise noted. All values shown as max or min on absolute value basis.

## Schematic Diagrams (DS8908 AM/FM PLL Typical Input/Output Schematics)





# Timing Diagrams\*



\*Timing diagrams are not drawn to scale. Scale within any one drawing may not be consistent, and intervals are defined positive as drawn.

## Serial Data Entry into the DS8908

Serial information entry into the DS8908 is enabled by a low level on the ENABLE input. One binary bit is then accepted from the DATA input with each positive transition of the CLOCK input. The CLOCK input must be low for the specified time preceding and following the negative transition of the ENABLE input.

The first two bits accepted following the negative transition of the ENABLE input are interpreted as address. If these address bits are *not* 1,1, *no* further information will be accepted from the DATA inputs, and the internal data latches *will not* be changed when ENABLE returns high.

If these first two bits *are* 1,1, then all succeeding bits are *accepted* as data, and are shifted successively into the internal shift register as long as ENABLE remains low.

Any data bits preceding the 19th to last bit will be shifted out, and thus are irrelevant. Data bits are counted as any bits *following* two valid address bits (1,1) with the ENABLE low. When the ENABLE input returns high, any further serial data entry is inhibited. Upon this positive transition, the data in the internal shift register is transferred into the internal data latches. Note that until this time, the states of the internal data latches have remained unchanged.

These data bits are interpreted as follows:

Data Bit Position	Data Interpretation
Last	Bit 19 Output (Pin 2)
2nd to Last	Bit 18 Output (Pin 1)
3rd to Last	Ref. Freq. Select Bit <sup>(1)</sup> 17
4th to Last	Ref. Freq. Select Bit <sup>(1)</sup> 16
5th to Last	AM/FM Select Bit
6th to Last	$(2^{13})$
7th to Last	$(2^{12})$
8th to Last	$(2^{11})$
9th to Last	$(2^{10})$
10th to Last	$(2^9)$
11th to Last	$(2^8)$
12th to Last	$(2^7)$
13th to Last	$(2^6)$
14th to Last	$(2^5)$
15th to Last	$(2^4)$
16th to Last	$(2^3)$
17th to Last	$(2^2)$
18th to Last	$(2^1)$
19th to Last	LSB of +N $(2^0)$

} ÷ N<sup>(2)</sup>

Note 1: See Reference Frequency Select Truth Table.

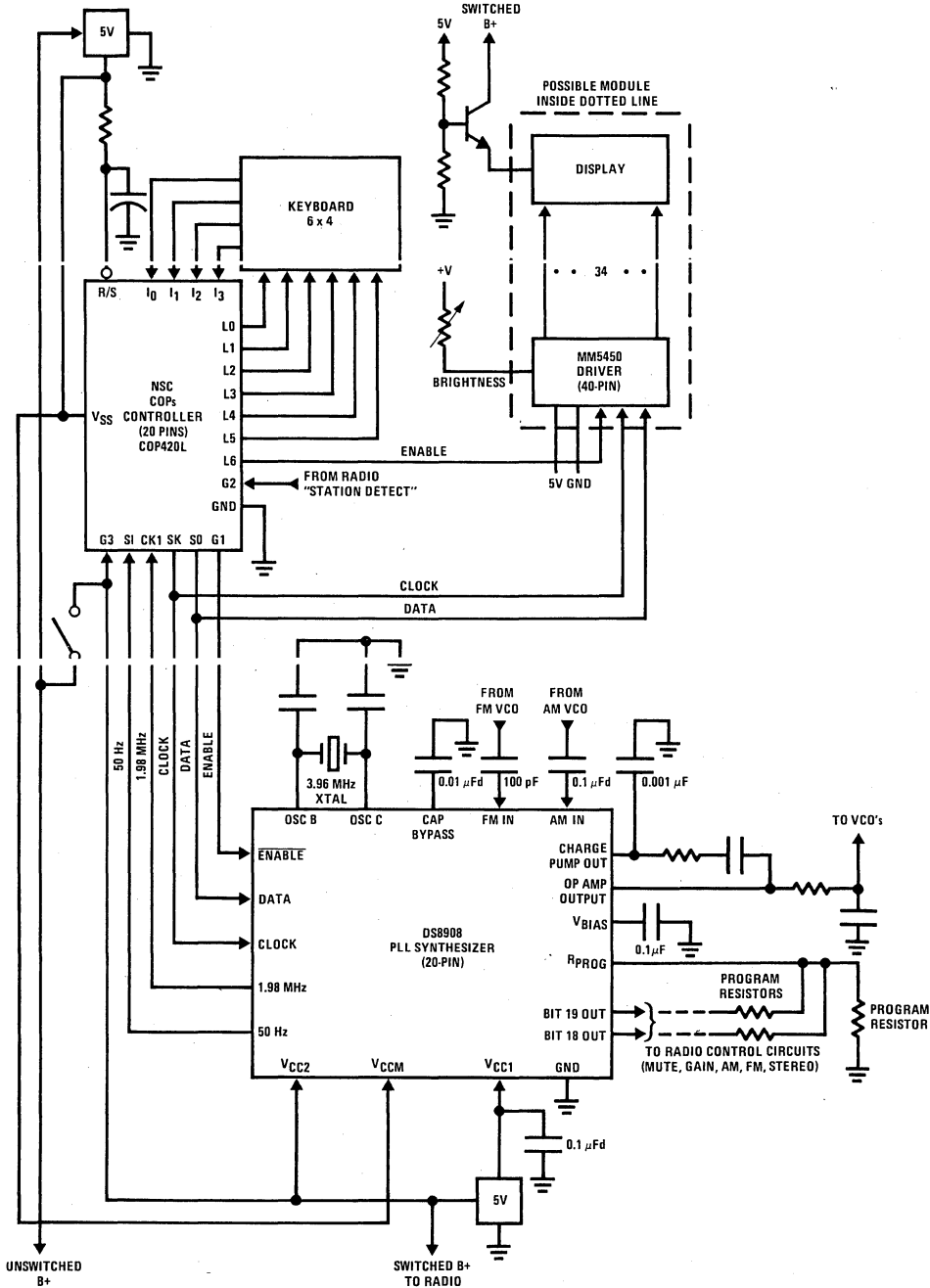
Note 2: The actual divide code is N + 1, i.e., the number loaded plus 1.

REFERENCE FREQUENCY SELECTION TRUTH TABLE

Serial Data		Reference Frequency
Bit 16	Bit 17	(kHz)
1	1	20
1	0	10
0	1	9
0	0	1

Typical Application

Electronically Tuned Radio Controller System; Direct Drive LED







# MM5445, MM5446, MM5447, MM5448 VF Display Drivers

## General Description

The MM5445 through MM5448 are monolithic MOS integrated circuits utilizing P-channel metal gate low threshold, enhancement mode and ion-implanted depletion mode devices. They are available in 40-pin molded dual-in-line packages. Each output can source up to 500  $\mu$ A at 2.0V maximum output voltage. A single pin controls the VF display brightness by setting the positive output voltage level.

- Wide power supply operation
- TTL compatibility
- 33, 34 or 35 outputs, 500  $\mu$ A source capability
- Alphanumeric capability
- Input data format compatible with MM5450, MM5451 LED drivers and MM5452, MM5453 LCD drivers

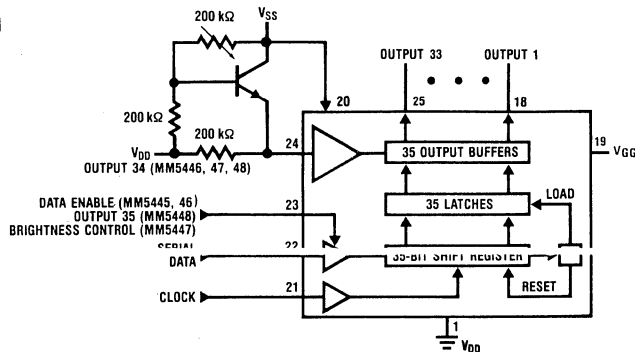
## Features

- Continuous brightness control
- Serial data input
- No load signal required
- Enable (on MM5445 and MM5446)

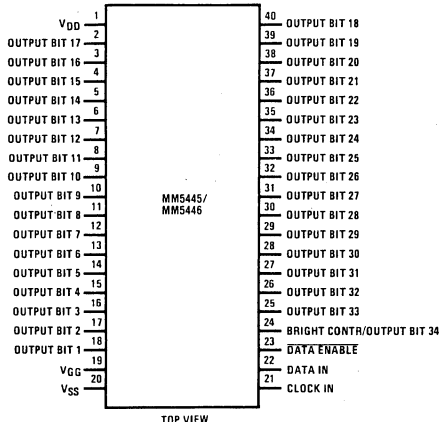
## Applications

- COPS or microprocessor displays
- Industrial control indicator
- Digital clock, thermometer, counter, voltmeter
- Instrumentation readouts

## Block Diagram

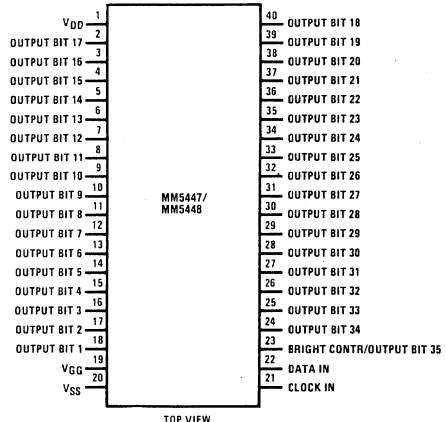


## Connection Diagrams (Dual-In-Line Packages)



Order Number MM5445N, MM5446N  
NS Package N40A

Figure 2a



Order Number MM5447N, MM5448N  
NS Package N40A

Figure 2b

## Absolute Maximum Ratings

Voltage at Any Pin	$V_{SS}$ to $V_{SS} - 30V$
Operating Temperature	$-40^{\circ}C$ to $+85^{\circ}C$
Storage Temperature	$-65^{\circ}C$ to $+150^{\circ}C$
Power Dissipation	560mW at $+85^{\circ}C$ 1W at $+25^{\circ}C$
Junction Temperature	$+150^{\circ}C$
Lead Temperature (Soldering, 10 seconds)	$300^{\circ}C$

## Electrical Characteristics

$T_A$  within operating range,  $V_{DD} = 0V$ ,  $V_{SS} = 4.5$  to  $5.5V$ , unless otherwise specified.

Parameter	Conditions	Min.	Typ.	Max.	Units
Power Supply					
$V_{SS}$		4.5	5.0	5.5	V
$V_{GG}$	$V_{SS} = 5V$	-25		-7	V
$V_{SS}$	$V_{DD} = V_{GG} = 0$	12		18	V
Power Supply Current					
$I_{SS}$	$V_{SS} = 5V$ , $V_{GG} = -25V$			9	mA
$I_{GG}$	$V_{DD} = 0$	-2			mA
Brightness Control	With respect to $V_{SS}$	$(V_{SS} - V_{GG})/2$		$V_{SS}$	V
Input Logic Levels					
Logic "0" Level	$-25V \leq V_{GG} \leq -7V$	-0.3		0.7	V
Logic "1" Level	$-25V \leq V_{GG} \leq -7V$	2.2		$V_{SS} + 0.3$	V
Logic "0" Level	$V_{DD} = V_{GG} = 0$	-0.3		1	V
Logic "1" Level	$V_{DD} = V_{GG} = 0$	$V_{SS} - 1$		$V_{SS} + 0.3$	V
Input Currents					
DATA IN and CLOCK		-10		10	$\mu A$
DATA ENABLE		-10		35	$\mu A$
BRIGHTNESS CONTROL	Excluding Output Loads (Note 2)			2	mA
Output Source Current					
Segment OFF	$V_{OUT} = (V_{SS} - V_{GG})/2$			-2	$\mu A$
Segment ON	$V_{OUT} = V_{SS} - 2V$ (Notes 1 and 2)	500			$\mu A$
Input Clock Frequency		0		250	kHz
Duty Cycle		40	50	60	%
Output Matching	$I_{OUT} = 500\mu A$	-0.5		0.5	V

**Note 1:** With Brightness Control tied to  $V_{SS}$  (MM5445 and MM5447) and  $V_{GG} = -25V$ .

**Note 2:** All output source current is provided from the Brightness Control input pin (MM5445 and MM5447).

## Functional Description

The MM5445 Series are specifically designed to operate 4 or 5-digit alphanumeric displays with minimal interface with the display and the data source. Character generation is done external to the MM5445 Series. Serial data transfer from the data source to the display driver is accomplished with 2 signals, serial data and clock. Using a format of a leading "1" followed by the 35 data bits allows data transfer without an additional load signal. The 35 data bits are latched after the 36th bit is complete, thus providing non-multiplexed, direct drive to the display. Outputs change only if the serial data bits differ from the previous time. Display brightness is determined by control of the positive output voltage level.

A block diagram is shown in *Figure 1*.

*Figure 2* shows the pin-out of the MM5445 series. Bit 1 is the first bit following the start bit and it will appear on pin 18. A logical "1" at the input will turn on the appropriate VF display segment.

*Figure 4* shows the input data format. A start bit of logical "1" precedes the 35 bits of data. At the 36th clock a LOAD signal is generated synchronously with the high state of the clock, which loads the 35 bits of the shift registers into the latches. At the low state of the clock a RESET signal is generated which clears all the shift registers for the next set of data. The shift registers are static master-slave configuration. There is no clear for the master portion of the first shift register, thus allowing continuous operation.

There must be a complete set of 36 clocks or the shift registers will not clear.

When the chip first powers ON an internal power ON reset signal is generated which resets all registers and all latches. The START bit and the first clock return the chip to its normal operation.

*Figure 3* shows the timing relationships between data, clock and data enable. A maximum clock frequency of 250 kHz is assumed.

# Typical Applications

MM5445, MM5446, MM5447, MM5448

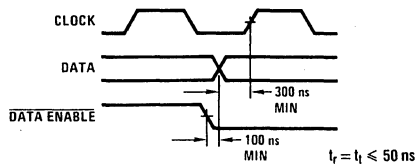


Figure 3

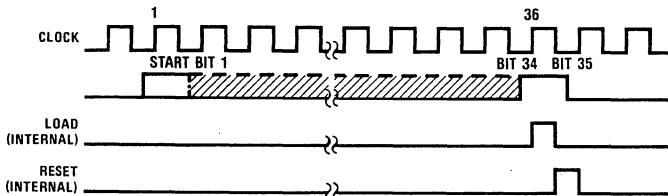
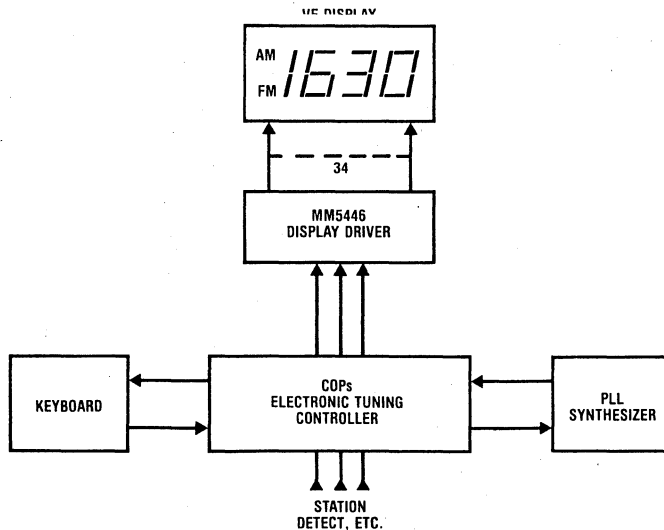


Figure 4. Input Data Format



Basic Electronically Tuned Radio System

5



# MM5450, MM5451 LED Display Drivers

## General Description

The 5450 and MM5451 are monolithic MOS integrated circuits utilizing N-channel metal-gate low threshold, enhancement mode, and ion-implanted depletion mode devices. They are available in 40-pin molded dual-in-line packages. A single pin controls the LED display brightness by setting a reference current through a variable resistor connected to  $V_{DD}$ .

- Enable (on MM5450)
- Wide power supply operation
- TTL compatibility
- 34 or 35 outputs, 15 mA sink capability
- Alphanumeric capability

## Features

- Continuous brightness control
- Serial data input
- No load signal required

## Applications

- COPS or microprocessor displays
- Industrial control indicator
- Relay driver
- Digital clock, thermometer, counter, voltmeter
- Instrumentation readouts

## Block Diagram

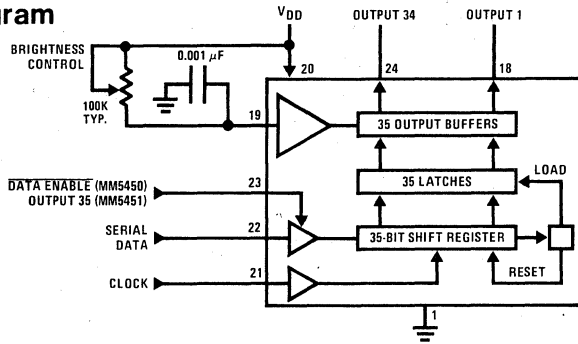
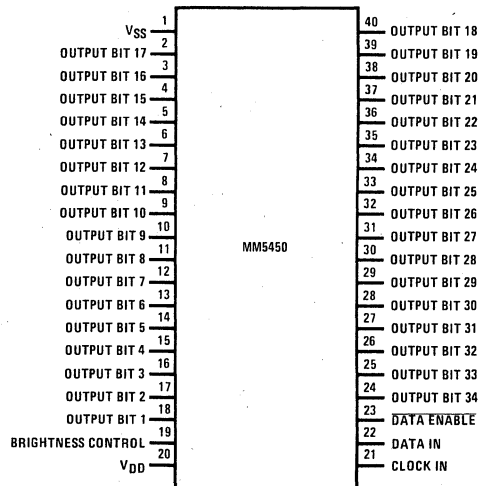


FIGURE 1

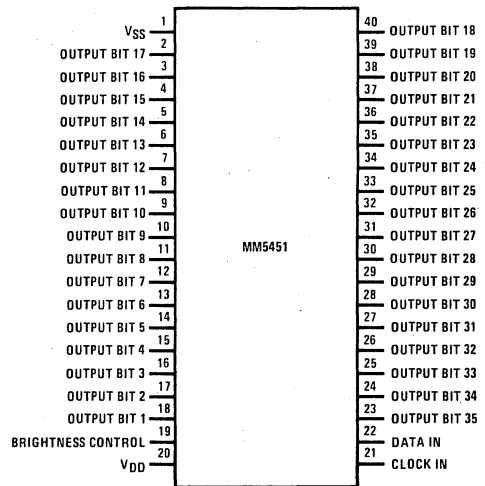
## Connection Diagrams (Dual-In-Line Packages)



TOP VIEW

Order Number MM5450N, MM5451N  
NS Package N40A

FIGURE 2a



TOP VIEW

Order Number MM5450D, MM5451D  
NS Package D40C

FIGURE 2b

## Absolute Maximum Ratings

Voltage at Any Pin	$V_{SS}$ to $V_{SS} + 12V$
Operating Temperature	$-25^{\circ}C$ to $+85^{\circ}C$
Storage Temperature	$-65^{\circ}C$ to $+150^{\circ}C$
Power Dissipation	560 mW at $+85^{\circ}C$ 1W at $+25^{\circ}C$
Junction Temperature	$+150^{\circ}C$
Lead Temperature (Soldering, 10 seconds)	$300^{\circ}C$

## Electrical Characteristics

$T_A$  within operating range,  $V_{DD} = 4.75V$  to  $11.0V$ ,  $V_{SS} = 0V$ , unless otherwise specified.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
Power Supply		4.75		11	V
Power Supply Current	Excluding Output Loads			7	mA
Input Voltages					
Logical "0" Level	$\pm 10 \mu A$ Input Bias	-0.3		0.8	V
Logical "1" Level	$4.75 \leq V_{DD} \leq 5.25$	2.2		$V_{DD}$	V
	$V_{DD} > 5.25$	$V_{DD} - 2$		$V_{DD}$	V
Brightness Input (Note 2)		0		0.75	mA
Output Sink Current (Note 3)					
Segment OFF	$V_{OUT} = 3.0V$			10	$\mu A$
Segment ON	$V_{OUT} = 1V$ (Note 4)				
	Brightness Input = $0 \mu A$	0		10	$\mu A$
	Brightness Input = $100 \mu A$	2.0	2.7	4	mA
	Brightness Input = $750 \mu A$	15		25	mA
Brightness Input Voltage (Pin 19)	Input Current = $750 \mu A$	3.0		4.3	V
Input Clock Frequency		0		0.5	MHz
Duty Cycle		40	50	60	%
Output Matching (Note 1)				$\pm 20$	%

**Note 1:** Output matching is calculated as the percent variation from  $I_{MAX} + I_{MIN}/2$ .

**Note 2:** With a fixed resistor on the brightness input pin some variation in brightness will occur from one device to another.

**Note 3:** Absolute maximum for each output should be limited to 40mA.

**Note 4:** The  $V_{OUT}$  voltage should be regulated by the user. See Figures 6 and 7 for allowable  $V_{OUT}$  vs.  $I_{OUT}$  operation.

## Functional Description

Both the MM5450 and the MM5451 are specifically designed to operate 4 or 5-digit alphanumeric displays with minimal interface with the display and the data source. Serial data transfer from the data source to the display driver is accomplished with 2 signals, serial data and clock. Using a format of a leading "1" followed by the 35 data bits allows data transfer without an additional load signal. The 35 data bits are latched after the 36th bit is complete, thus providing non-multiplexed, direct drive to the display. Outputs change only if the serial data bits differ from the previous time. Display brightness is determined by control of the output current for LED displays. A 0.001 capacitor should be connected to brightness control, pin 19, to prevent possible oscillations.

A block diagram is shown in Figure 1. For the MM5450 a DATA ENABLE is used instead of the 35th output. The DATA ENABLE input is a metal option for the MM5450. The output current is typically 20 times greater than the current into pin 19, which is set by an external variable resistor. There is an internal limiting resistor of  $400\Omega$  nominal value.

Figure 4 shows the input data format. A start bit of logical "1" precedes the 35 bits of data. At the 36th clock a LOAD signal is generated synchronously with the high state of the clock, which loads the 35 bits of the shift registers into the latches. At the low state of the clock a RESET signal is generated which clears all the shift registers for the next set of data. The shift registers are static master-slave configuration. There is no clear for the master portion of the first shift register, thus allowing continuous operation.

There must be a complete set of 36 clocks or the shift registers will not clear.

When the chip first powers ON an internal power ON reset signal is generated which resets all registers and all latches. The START bit and the first clock return the chip to its normal operation.

Figure 2 shows the pin-out of the MM5450 and MM5451. Bit 1 is the first bit following the start bit and it will appear on pin 18. A logical "1" at the input will turn on the appropriate LED.

## Functional Description (Continued)

Figure 3 shows the timing relationships between data, clock and data enable. A max clock frequency of 0.5 MHz is assumed.

For applications where a lesser number of outputs are used, it is possible to either increase the current per output, or operate the part at higher than 1V  $V_{OUT}$ . The following equation can be used for calculations.

$$T_j = (V_{OUT}) (I_{LED}) (\text{No. of segments}) (124^\circ\text{C/W}) + T_A$$

where:

$T_j$  = junction temperature +150°C max

$V_{OUT}$  = the voltage at the LED driver outputs

$I_{LED}$  = the LED current

124°C/W = thermal coefficient of the package

$T_A$  = ambient temperature

The above equation was used to plot Figure 5, Figure 6, and Figure 7.

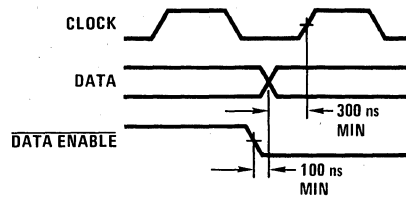


FIGURE 3

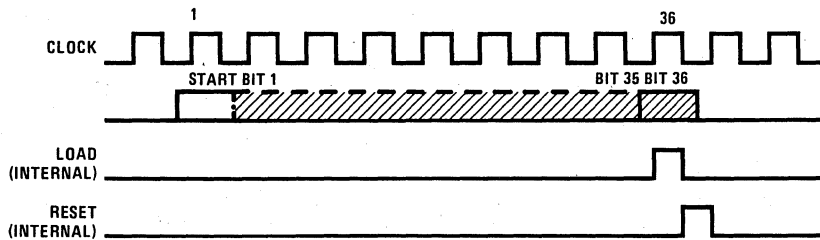


FIGURE 4. Input Data Format

# Typical Applications

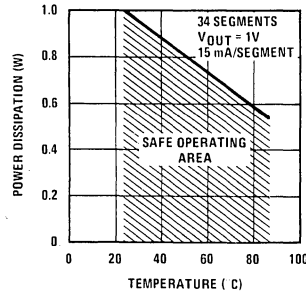


FIGURE 5

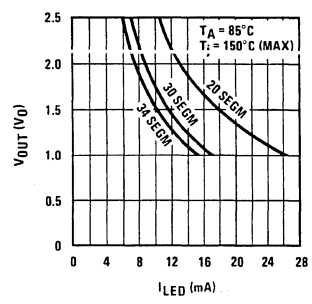


FIGURE 6

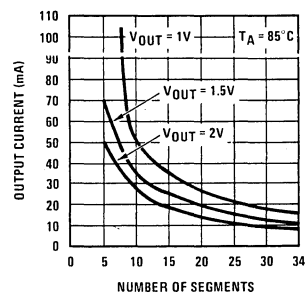
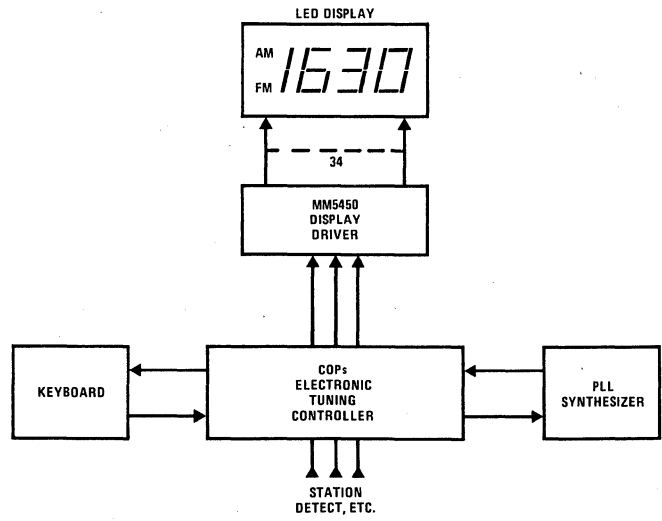


FIGURE 7

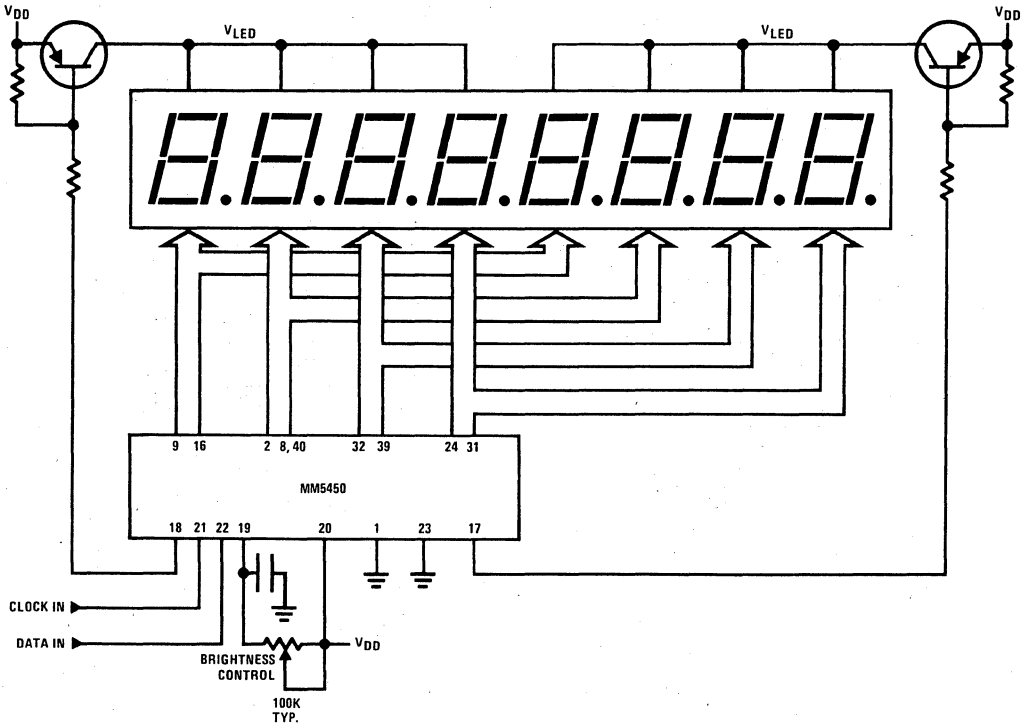
## Basic Electronically Tuned Radio System





# Typical Applications (Continued)

## Duplexing 8 Digits with One MM5450





# MM5452, MM5453 Liquid Crystal Display Drivers

## General Description

The MM5452 is a monolithic integrated circuit utilizing CMOS metal gate, low threshold enhancement mode devices. It is available in a 40-pin molded package. The chip can drive up to 32 segments of LCD and can be paralleled to increase this number. The chip is capable of driving a 4 1/2-digit 7-segment display with minimal interface between the display and the data source.

The MM5452 stores the display data in latches after it is clocked in, and holds the data until new display data is received.

- DATA ENABLE (MM5452)
- Wide power supply operation
- TTL compatibility
- 32 or 33 outputs
- Alphanumeric and bar graph capability
- Cascaded operation capability

## Applications

- COPs or microprocessor displays
- Industrial control indicator
- Digital clock, thermometer, counter, voltmeter
- Instrumentation readouts
- Remote displays

## Features

- Serial data input
- No load signal required

## Block and Connection Diagrams

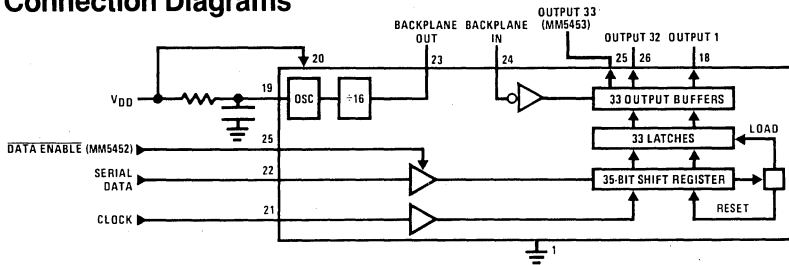


FIGURE 1

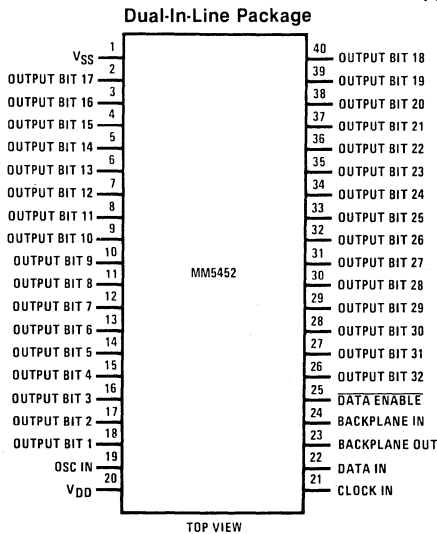


FIGURE 2a

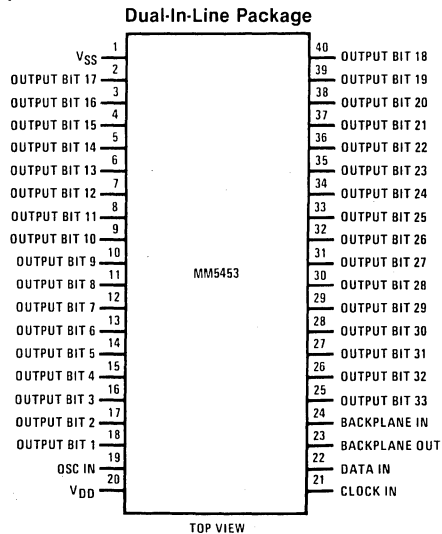


FIGURE 2b

Order Number MM5452N, MM5453N  
NS Package N40A

## Absolute Maximum Ratings

Voltage at Any Pin	$V_{SS}$ to $V_{SS} + 10V$	Power Dissipation	300 mW at +85°C
Operating Temperature	-40°C to +85°C		350 mW at +25°C
Storage Temperature	-65°C to +150°C	Junction Temperature	+150°C
		Lead Temperature (Soldering, 10 seconds)	300°C

## Electrical Characteristics

$T_A$  within operating range,  $V_{DD} = 3.0V$  to  $10V$ ,  $V_{SS} = 0V$ , unless otherwise specified.

Parameter	Conditions	Min	Typ	Max	Units
Power Supply		3		10	V
Power Supply Current	Excluding Outputs			40	$\mu A$
	OSC = $V_{SS}$ , BP IN @ 32 Hz $V_{DD} = 5V$ , Open Outputs, No Clock			10	$\mu A$
Clock Frequency				500	kHz
Input Voltages	Logical '0' Level	$V_{DD} < 4.75$	-0.3	$0.1 V_{DD}$	V
		$V_{DD} \geq 4.75$	-0.3	0.8	V
Logical '1' Level		$V_{DD} > 5.25$	$0.9 V_{DD}$	$V_{DD}$	V
		$V_{DD} \leq 5.25$	2.0	$V_{DD}$	V
Output Current Levels	Segments				
	Sink	$V_{DD} = 3V$ , $V_{OUT} = 0.3V$		-20	$\mu A$
Source	$V_{DD} = 3V$ , $V_{OUT} = V_{DD} - 0.3V$	20			$\mu A$
Backplane	Sink	$V_{DD} = 3V$ , $V_{OUT} = 0.3V$		-320	$\mu A$
	Source	$V_{DD} = 3V$ , $V_{OUT} = V_{DD} - 0.3V$	320		$\mu A$
Output Offset Voltage	Segment Load 250 pF Backplane Load 8750 pF			$\pm 50$	mV

## Functional Description (Continued)

The MM5452 is specifically designed to operate 4 1/2-digit, 7-segment displays with minimal interface with the display and the data source. Serial data transfer from the data source to the display driver is accomplished with 2 signals, serial data and clock. Since the MM5452 does not contain a character generator, the formatting of the segment information must be done prior to inputting the data to the MM5452. Using a format of a leading "1" followed by the 32 data bits allows data transfer without an additional load signal. The 32 data bits are latched after the 36th clock is complete, thus providing non-multiplexed, direct drive to the display. Outputs change only if the serial data bits differ from the previous time.

A block diagram is shown in Figure 1. For the MM5452 a DATA ENABLE is used instead of the 33rd output. If the DATA ENABLE signal is not required, the 33rd output can be brought out. This is the MM5453 device.

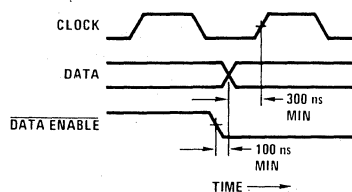


FIGURE 3

Figure 4 shows the input data format. A start bit of logical "1" precedes the 32 bits of data. At the 36th clock a LOAD signal is generated synchronously with the high state of the clock, which loads the 32 bits of the shift registers into the latches. At the low state of the clock a RESET signal is generated which clears all the shift registers for the next set of data. The shift registers are static master-slave configuration. There is no clear for the master portion of the first shift register, thus allowing continuous operation.

If the clock is not continuous, there must be a complete set of 36 clocks otherwise the shift registers will not clear.

Figure 2a shows the pin-out of the MM5452. Bit 1 is the first bit following the start bit and it will appear on pin 18.

Figure 3 shows the timing relationships between data, clock and DATA ENABLE.

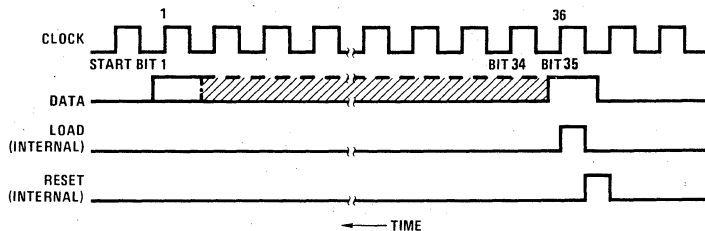


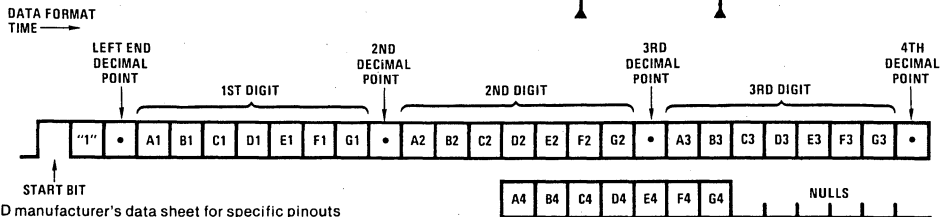
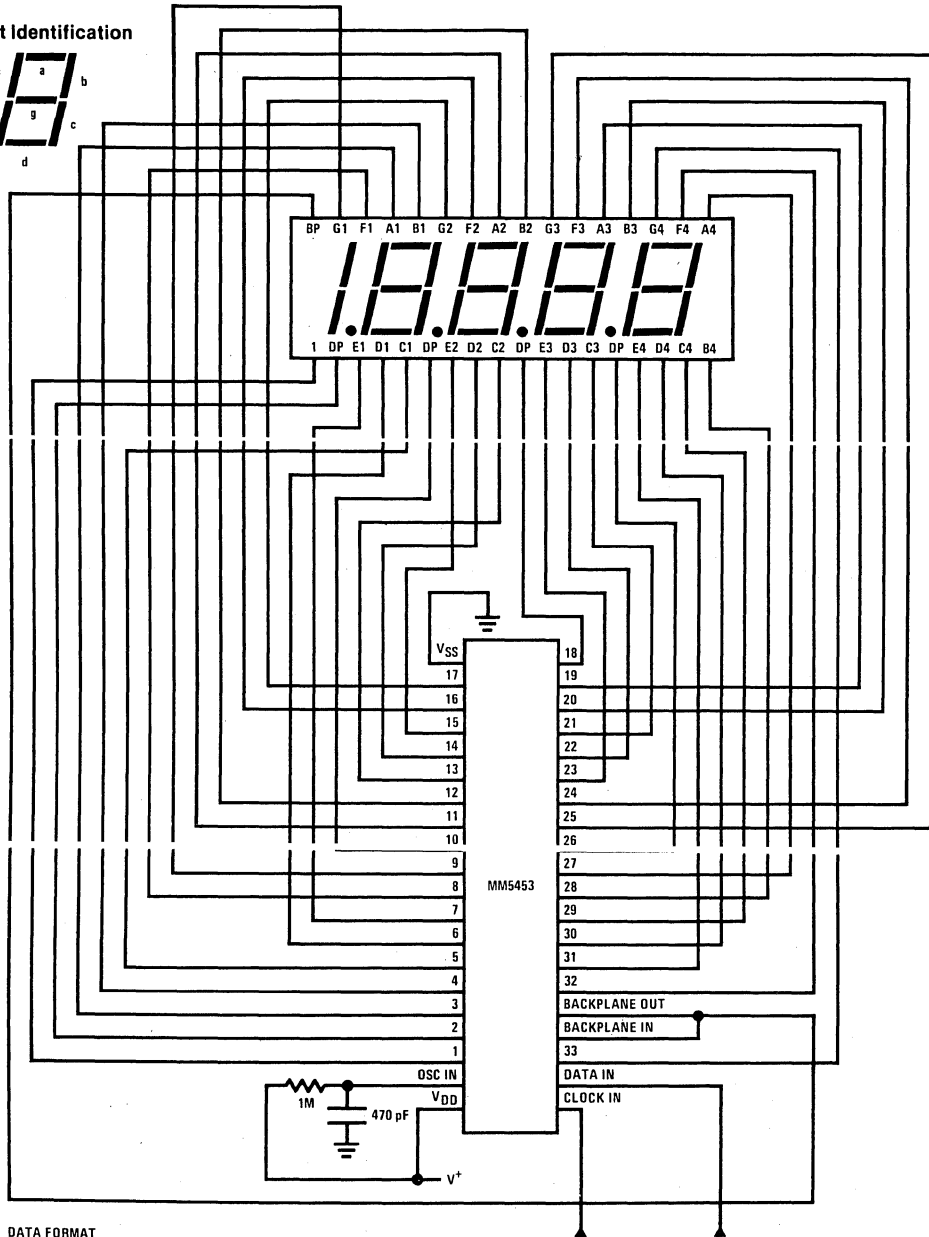
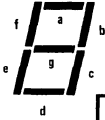
FIGURE 4. Input Data Format

# Functional Description (Continued)

Figure 5 shows a typical application. Note how the input data maps to the output pins and the display. The MM5452 and MM5453 do not have format restrictions, as all outputs

are controllable. This application assumes a specific display pinout. Different display/driver connection patterns will, of course, yield a different input data format.

## Segment Identification



Consult LCD manufacturer's data sheet for specific pinouts

FIGURE 5. Typical 4 1/2-Digit Display Application

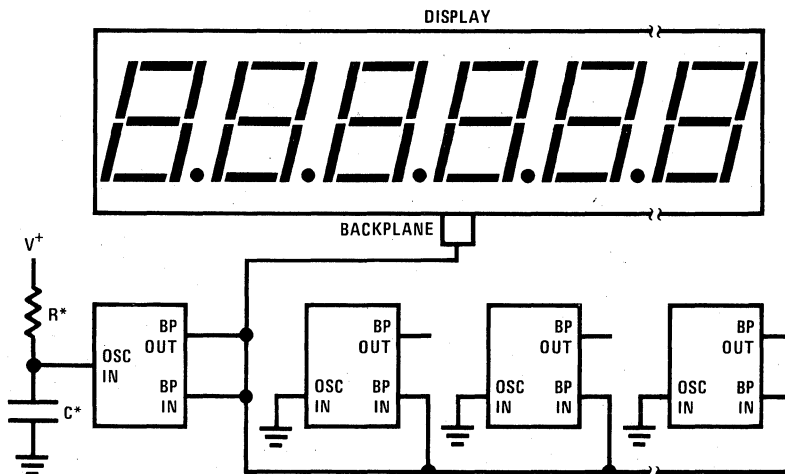
## Functional Description (Continued)

Figure 8 shows a four wire remote display that takes advantage of the device's serial input to move many bits of display information on a few wires.

### Using an External Clock

The MM5452, MM5453 LCD Drivers can be used with an externally supplied clock, provided it has a duty cycle of 50%.

Deviations from a 50% duty cycle result in an offset voltage on the LCD. In Figure 7, a flip flop is used to assure a 50% duty cycle. The oscillator input is grounded to prevent oscillation and reduce current consumption in the chips. The oscillator is not used.



\* The minimum recommended value for R for the oscillator input is 9 k $\Omega$ . An RC time constant of approximately  $4.91 \times 10^{-4}$  should produce a backplane frequency between 30 Hz and 150 Hz.

FIGURE 6. Parallel Backplane Outputs

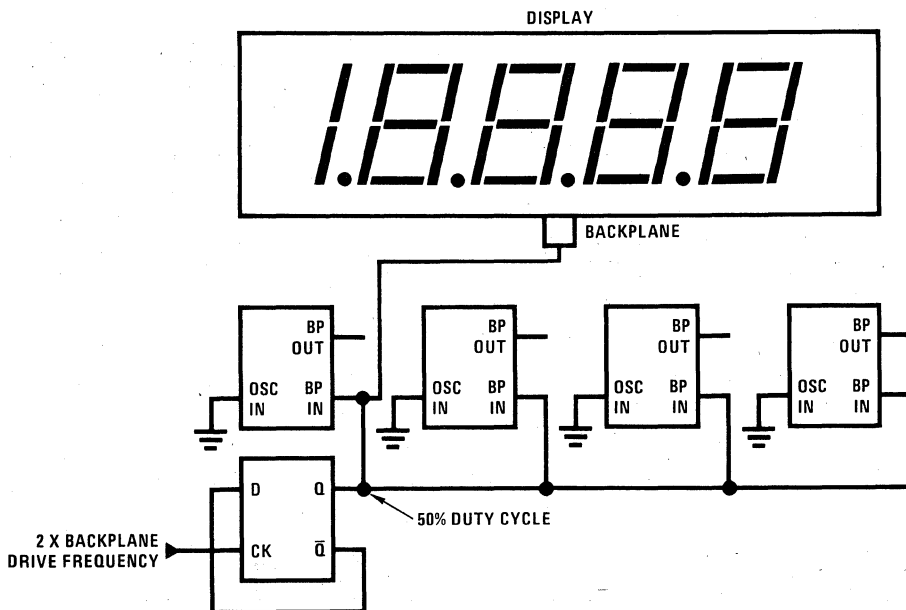


FIGURE 7. External Backplane Clock

## Functional Description (Continued)

Using an external clock allows synchronizing the display drive with AC power, internal clocks, or DVM integration time to reduce interference from the display.

Figure 9 is a general block diagram that shows how the device's serial input can be used to advantage in an analog display. The analog voltage input is compared with a staircase voltage generated by a counter and a digital-to-analog converter. The result of this comparison is clocked into the MM5452, MM5453.

The next clock pulse increments the staircase and clocks the new data in.

With a buffer amplifier, the same staircase waveform can be used for many displays. The digital-to-analog converter need not be linear; logarithmic or other non-linear functions can be displayed by using weighted resistors or special DACs. This system can be used for status indicators, spectrum analyzers, audio level and power meters, tuning indicators, and other applications.

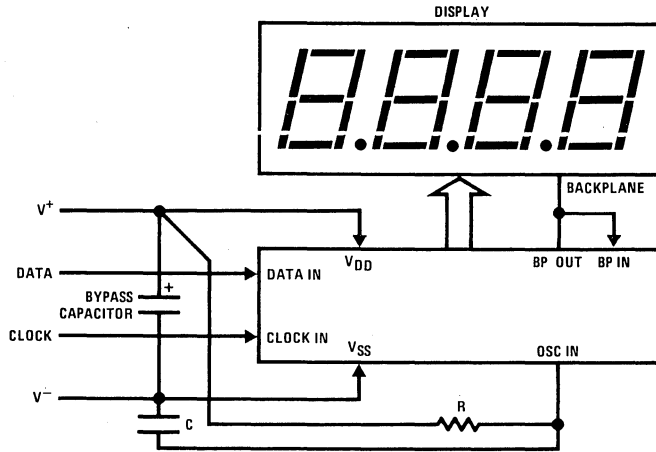
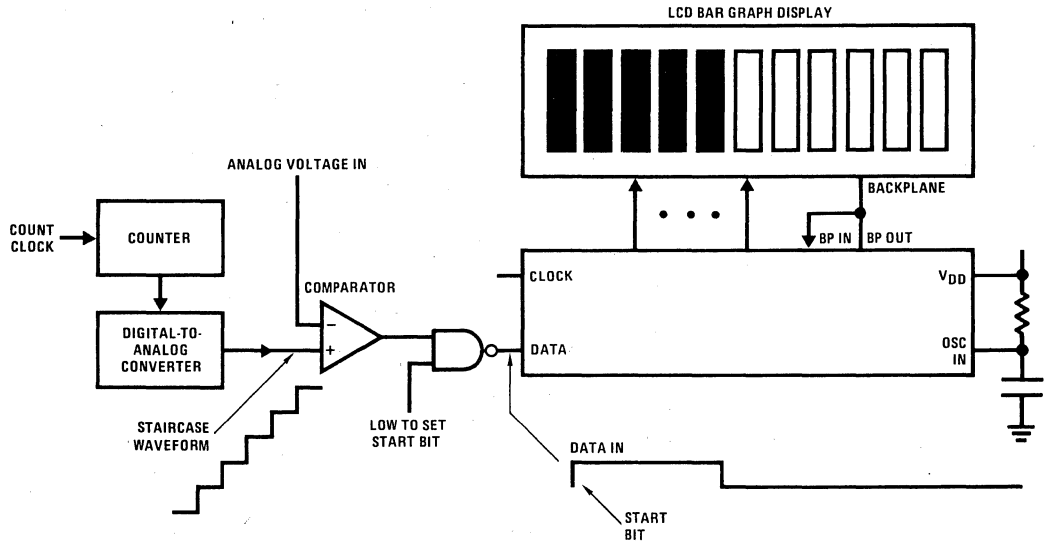


FIGURE 8. Four Wire Remote Display



Data is high until staircase > input

FIGURE 9. Analog Display

## MM5480 LED Display Driver

### General Description

The 5480 is a monolithic MOS integrated circuit utilizing N-channel metal gate low threshold, enhancement mode and ion-implanted depletion mode devices. It utilizes the MM5451 die packaged in a 28-pin package making it ideal for a 3½ digit display. A single pin controls the LED display brightness by setting a reference current through a variable resistor connected either to  $V_{DD}$  or to a separate supply of 11V maximum.

- Wide power supply operation
- TTL compatibility
- Alphanumeric capability
- 3½ digit displays

### Features

- Continuous brightness control
- Serial data input
- No load signal required

### Applications

- COPS or microprocessor displays
- Industrial control indicator
- Relay driver
- Digital clock, thermometer, counter, voltmeter
- Instrumentation readouts

### Block Diagram

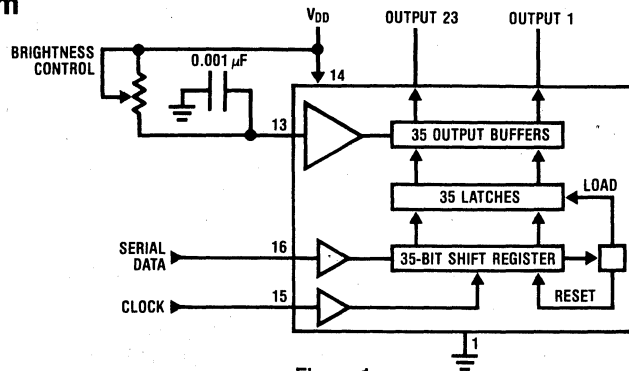
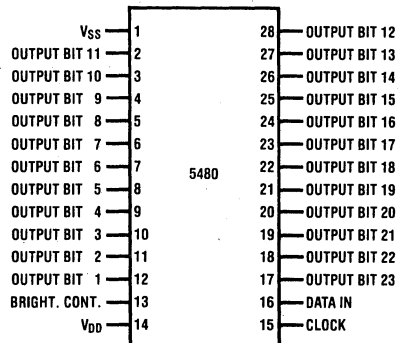


Figure 1

### Connection Diagram (Dual-In-Line Packages)



Order Number MM5480N  
NS Package N28A

Figure 2

## Absolute Maximum Ratings

Voltage at Any Pin	$V_{SS}$ to $V_{SS} + 12V$
Operating Temperature	$-25^{\circ}C$ to $+85^{\circ}C$
Storage Temperature	$-65^{\circ}C$ to $+150^{\circ}C$
Power Dissipation	490 mW at $+85^{\circ}C$ 940 mW at $+25^{\circ}C$
Junction Temperature	$+150^{\circ}C$
Lead Temperature (Soldering, 10 seconds)	$300^{\circ}C$

## Electrical Characteristics

$T_A$  within operating range,  $V_{DD} = 4.75$  to  $11.0V$ ,  $V_{SS} = 0V$ , unless otherwise specified.

Parameter	Conditions	Min.	Typ.	Max.	Units
Power Supply		4.75		11.0	V
Power Supply Current	Excluding Output Loads			7.0	mA
Input Voltages					
Logical "0" Level	$\pm 10\mu A$ Input Bias	-0.3		0.8	V
Logical "1" Level	$4.75 \leq V_{DD} \leq 5.25$	2.2		$V_{DD}$	V
	$V_{DD} > 5.25$	$V_{DD} - 2$		$V_{DD}$	V
Brightness Input (Note 2)		0		0.75	mA
Output Sink Current (Note 3)					
Segment OFF	$V_{OUT} = 3.0V$			10.0	$\mu A$
Segment ON	$V_{OUT} = 1V$ (Note 4)				
	Brightness Input = $0\mu A$	0		10.0	$\mu A$
	Brightness Input = $100\mu A$	2.0	2.7	4.0	mA
	Brightness Input = $750\mu A$	15.0		25.0	mA
Maximum Segment Current				40.0	mA
Brightness Input Voltage (Pin 13)	Input Current = $750\mu A$	3.0		4.3	V
Input Clock Frequency		0		0.5	MHz
Duty Cycle		40	50	60	%
Output Matching (Note 1)				$\pm 20$	%

**Note 1:** Output matching is calculated as the percent variation from  $I_{MAX} + I_{MIN}/2$ .

**Note 2:** With a fixed resistor on the brightness input pin some variation in brightness will occur from one device to another.

**Note 3:** Absolute maximum for each output should be limited to 40 mA

**Note 3:** The  $V_{OUT}$  voltage should be regulated by the user.

**Note 4:** The  $V_{OUT}$  voltage should be regulated by the user.

## Functional Description

The MM5480 is specifically designed to operate 3½-digit alphanumeric displays with minimal interface with the display and the data source. Serial data transfer from the data source to the display driver is accomplished with 2 signals, serial data and clock. Using a format of a leading "1" followed by the 35 data bits allows data transfer without an additional load signal. The 35 data bits are latched after the 36th bit is complete, thus providing non-multiplexed, direct drive to the display. Outputs change only if the serial data bits differ from the previous time. Display brightness is determined by control of the output current for LED displays. A 0.001 capacitor should be connected to brightness control, pin 13, to prevent possible oscillations.

A block diagram is shown in *Figure 1*. The output current is typically 20 times greater than the current into pin 13, which is set by an external variable resistor. There is an internal limiting resistor of 400 $\Omega$  nominal value.

*Figure 4* shows the input data format. A start bit of logical "1" precedes the 35 bits of data. At the 36th clock a LOAD signal is generated synchronously with the high state of the clock, which loads the 35 bits of the shift registers into the latches. At the low state of the clock a RESET signal is generated which clears all the shift reg-

isters for the next set of data. The shift registers are static master-slave configuration. There is no clear for the master portion of the first shift register, thus allowing continuous operation.

There must be a complete set of 36 clocks or the shift registers will not clear.

When the chip first powers ON an internal power ON reset signal is generated which resets all registers and all latches. The START bit and the first clock return the chip to its normal operation.

*Figure 5* shows the Output Data Format for the 5480. Because it uses only 23 of the possible 35 outputs, 12 of the bits are 'Don't Cares'.

*Figure 3* shows the timing relationships between data, clock, and data enable. A maximum clock frequency of 0.5 MHz is assumed.

For applications where a lesser number of outputs are used, it is possible to either increase the current per output, or operate the part at higher than 1V  $V_{OUT}$ . The following equation can be used for calculations.

$$T_j = (V_{OUT}) (I_{LED}) (\text{No. of segments}) (132^{\circ}C/W) + T_A$$



# Functional Description (Continued)

where:

$T_j$  = junction temperature + 150°C max.

132°C/W = thermal coefficient of the package

$V_{OUT}$  = the voltage at the LED driver outputs

$T_A$  = ambient temperature

$I_{LED}$  = the LED current

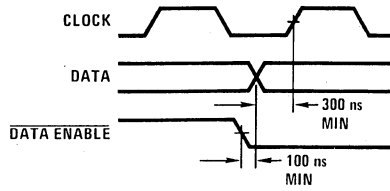


Figure 3

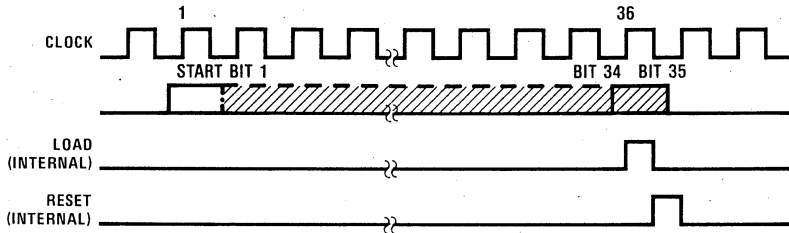
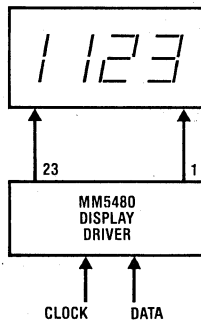


Figure 4. Input Data Format

5451	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	START		
5480	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	START

Figure 5. Output Data Format



Basic 3 1/2 Digit Interface

# MM5481 LED Display Driver

## General Description

The 5481 is a monolithic MOS integrated circuit utilizing N-channel metal gate low threshold, enhancement mode and ion-implanted depletion mode devices. It utilizes the MM5450 die packaged in a 20-pin package making it ideal for a 2 digit display. A single pin controls the LED display brightness by setting a reference current through a variable resistor connected either to  $V_{DD}$  or to a separate supply of 11V maximum.

- Wide power supply operation
- TTL compatibility
- Alphanumeric capability
- 2 digit LED driver

## Features

- Continuous brightness control
- Serial data input
- No load signal required
- Data enable

## Applications

- COPS or microprocessor displays
- Industrial control indicator
- Relay driver
- Instrumentation readouts

## Block Diagram

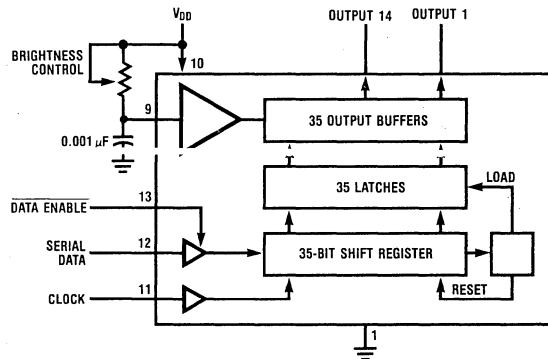
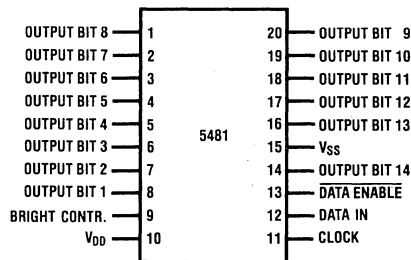


Figure 1

## Connection Diagram

(Dual-In-Line Package)



Top View

Order Number MM5481N  
NS Package N20A

Figure 2

## Absolute Maximum Ratings

Voltage at Any Pin	$V_{SS}$ to $V_{SS} + 12V$
Operating Temperature	-25°C to +85°C
Storage Temperature	-65°C to +150°C
Power Dissipation	450 mW at +85°C 860 mW at +25°C
Junction Temperature	+150°C
Lead Temperature (Soldering, 10 seconds)	300°C

## Electrical Characteristics

$T_A$  within operating range,  $V_{DD} = 4.75$  to  $11.0V$ ,  $V_{SS} = 0V$ , unless otherwise specified.

Parameter	Conditions	Min.	Typ.	Max.	Units
Power Supply		4.75		11	V
Power Supply Current	Excluding Output Loads			7	mA
Input Voltages					
Logical "0" Level	$\pm 10 \mu A$ Input Bias	-0.3		0.8	V
Logical "1" Level	$4.75 \leq V_{DD} \leq 5.25$	2.2		$V_{DD}$	V
	$V_{DD} > 5.25$	$V_{DD} - 2$		$V_{DD}$	V
Brightness Input (Note 2)		0		0.75	mA
Output Sink Current (Note 3)					
Segment OFF	$V_{OUT} = 3.0V$			10	$\mu A$
Segment ON	$V_{OUT} = 1V$ (Note 4)				
	Brightness Input = $0 \mu A$	0		10	$\mu A$
	Brightness Input = $100 \mu A$	2.0	2.7	4.0	mA
	Brightness Input = $750 \mu A$	15		25	mA
Maximum Segment Current				40	mA
Brightness Input Voltage (Pin 9)	Input Current = $750 \mu A$	3.0		4.3	V
Input Clock Frequency		0		0.5	MHz
Duty Cycle		40	50	60	%
Output Matching (Note 1)				$\pm 20$	%

**Note 1:** Output matching is calculated as the percent variation from  $I_{MAX} + I_{MIN}/2$ .

**Note 2:** With a fixed resistor on the brightness input pin some variation in brightness will occur from one device to another.

**Note 3:** Absolute maximum for each output should be limited to 40 mA

**Note 4:** The  $V_{OUT}$  voltage should be regulated by the user.

## Functional Description

The MM5481 uses the 5450 die which is packaged to operate 2-digit alphanumeric displays with minimal interface with the display and the data source. Serial data transfer from the data source to the display driver is accomplished with 2 signals, serial data and clock. Using a format of a leading "1" followed by the 35 data bits allows data transfer without an additional load signal. The 35 data bits are latched after the 36th bit is complete, thus providing non-multiplexed, direct drive to the display. Outputs change only if the serial data bits differ from the previous time. Display brightness is determined by control of the output current for LED displays. A 0.001 capacitor should be connected to brightness control, pin 9, to prevent possible oscillations.

A block diagram is shown in *Figure 1*. The output current is typically 20 times greater than the current into pin 9, which is set by an external variable resistor. There is an internal limiting resistor of 400 $\Omega$  nominal value.

*Figure 4* shows the input data format. A start bit of logical "1" precedes the 35 bits of data. At the 36th clock a LOAD signal is generated synchronously with the high state of the clock, which loads the 35 bits of the shift

registers into the latches. At the low state of the clock a RESET signal is generated which clears all the shift registers for the next set of data. The shift registers are static master-slave configuration. There is no clear for the master portion of the first shift register, thus allowing continuous operation.

There must be a complete set of 36 clocks or the shift registers will not clear.

When the chip first powers ON an internal power ON reset signal is generated which resets all registers and all latches. The START bit and the first clock return the chip to its normal operation.

*Figure 5* shows the Output Data Format for the 5481. Because it uses only 14 of the possible 34 outputs, 20 of the bits are 'Don't Cares'. Note that only alternate groups of 4 outputs are used.

*Figure 3* shows the timing relationships between data, clock, and data enable. A maximum clock frequency of 0.5 MHz is assumed.

### Functional Description (Continued)

For applications where a lesser number of outputs are used, it is possible to either increase the current per output, or operate the part at higher than 1V  $V_{OUT}$ . The following equation can be used for calculations.

$$T_j = (V_{OUT}) (I_{LED}) (\text{No. of segments}) (145^\circ\text{C/W}) + T_A$$

where:

- $T_j$  = junction temperature + 150°C max.
- $V_{OUT}$  = the voltage at the LED driver outputs
- $I_{LED}$  = the LED current
- 145°C/W = thermal coefficient of the package
- $T_A$  = ambient temperature

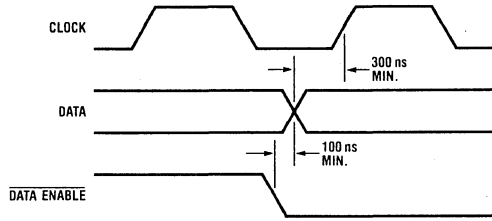


Figure 3

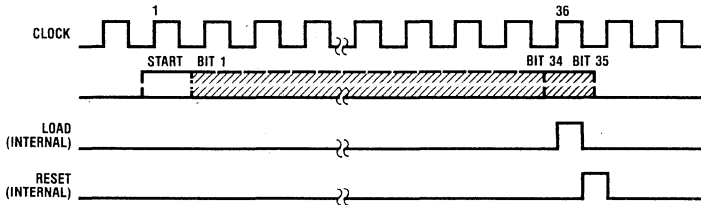
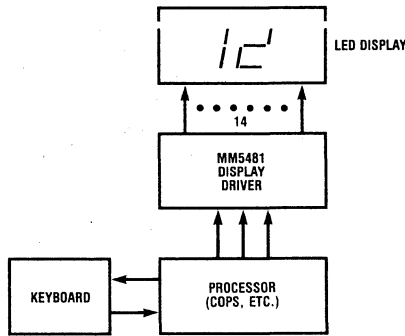


Figure 4. Input Data Format

5450	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	START	
5481	X	X	X	X	X	14	13	X	X	X	X	12	11	10	9	X	X	X	X	8	7	6	5	X	X	X	X	4	3	2	1	X	X	X	X	START

Figure 5. Output Data Format



Basic Electronically Tuned Television System



# MM5484, MM5485 16-, 11-Segment LED Display Drivers

## General Description

The MM5484, MM5485 are low threshold N-channel metal gate circuits using low threshold enhancement and ion implanted depletion devices. The MM5484 is available in a 22-pin molded package and is capable of driving 16 LED segments while the MM5485 is available in a 16-pin molded package and is capable of driving 11 LED segment outputs.

- TTL compatibility
- No load signal required
- Non multiplex display
- 2½ digit capability—MM5484
- 1½ digit capability—MM5485

## Features

- Serial data input
- Wide power supply operation
- 16 or 11 outputs, 15mA sink capability
- MM5484 is cascadeable

## Applications

- COPSTM or microprocessor displays
- Instrumentation readouts
- Industrial control indicator
- Relay driver

COPS is a trademark of National Semiconductor Corp.

## Block Diagrams

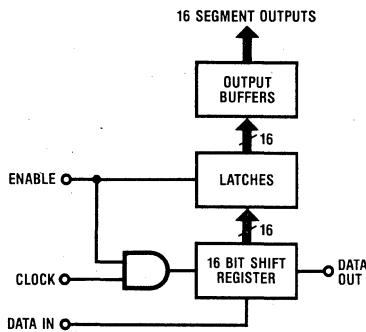


Figure 1. MM5484

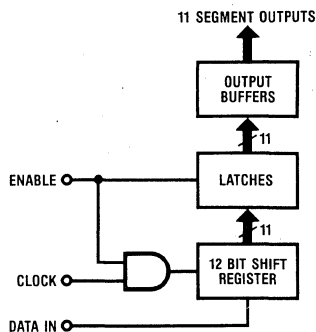
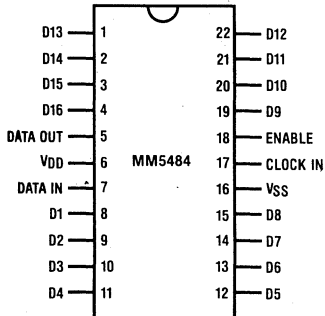
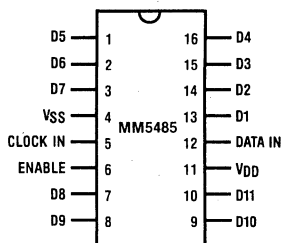


Figure 2. MM5485

## Connection Diagrams (Top Views)



Order Number MM5484N  
NS Package Number N22A



Order Number MM5485N  
NS Package Number N16A

Figure 4.

## Absolute Maximum Ratings

Voltage at LED outputs	$V_{SS} - 0.5V$ to $V_{SS} + 12V$
Voltage at other pins	$V_{SS} - 0.5V$ to $V_{SS} + 10V$
Operating Temperature	$-40^{\circ}C$ to $85^{\circ}C$
Storage Temperature	$-40^{\circ}C$ to $150^{\circ}C$
Lead Temperature (Soldering, 10 seconds)	$300^{\circ}C$
Maximum Power Dissipation	
MM5484	500 mW
MM5485	400 mW

## DC Electrical Characteristics $V_{DD} = 4.5$ to $9V$ , $T_A = -40^{\circ}C$ to $85^{\circ}C$ unless otherwise specified

Parameter	Conditions	Min.	Typ.	Max.	Units
Supply Voltage		4.5		9	V
Supply Current			5	10	mA
Logic One					
Input High Level $V_{IH}$		2.4		$V_{DD} + 0.5$	V
Logic Zero					
Input Low Level $V_{IL}$		0		0.8	V
Input Current	High or Low Level			$\pm 1$	$\mu A$
Input Capacitance				7.5	pF
Outputs					
Data Output Voltage	(Only for MM5484)				
High Level $V_{OH}$	$I_{OUT} = 0.1 mA$	$V_{DD} - 0.5$			V
Low Level $V_{OL}$	$I_{OUT} = -0.1 mA$			0.5	V
Segment Off (logic zero on input)	$V_{OUT} = 12V$ $R_{EXT} = 400 \Omega$			50	$\mu A$
Output Current					
Segment On (logic one on input)					
Output Voltage	$I_{OUT} = 15 mA$ $V_{DD} \geq 6V$		0.5	1.0	V

**Note 1:** Under no condition should the power dissipated by the segment driver exceed 50mW nor the entire chip power dissipation exceed 500mW for the MM5484 and 400mW for the MM5485.

## AC Electrical Characteristics (See Figure 3.) $V_{DD} = 4.5$ to $9V$ , $T_A = -40^{\circ}C$ to $85^{\circ}C$ unless otherwise specified

Symbol	Parameter	Conditions	Min.	Typ.	Max.	Units
	Clock Frequency				1	MHz
$t_{S1}$	Data Setup Time		0.5			$\mu s$
$t_{H1}$	Data Hold Time		0.5			$\mu s$
$t_{S2}$	Enable Setup Time		0.5			$\mu s$
$t_{H2}$	Enable Hold Time		0.5			$\mu s$
	Clock Rise Time				0.5	$\mu s$
	Data Out Delay				0.5	$\mu s$
tpd	Clock Period $t (= 1/f)$		2			$\mu s$

## Functional Description

The MM5484 and MM5485 are designed to drive LED displays directly. Serial data transfer from the data source to the display driver is accomplished with 3 signals, DATA IN, CLOCK and ENABLE. The signal ENABLE acts as an envelope and only while this signal is at a logic '1' do the circuits recognize the clock signal.

While ENABLE is high, data on the serial data input is transferred and shifted in the internal shift register on the rising clock edge, i.e. a logic '0' to logic '1' transition.

When the ENABLE signal goes to a low (logic zero state), the contents of the shift register is latched and the display will show the new data. While new data is being loaded into the SR the display will continue to show the old data.

For the MM5484, data is output from the serial DATA OUT pin on the falling edge of clock so cascading is made simple with race hazards eliminated.

The MM5485 is essentially a metal mask option of the MM5484 where only 11 segments are used. However, the MM5485 contains a 12-bit shift register and so when entering new data to this device 12 clock pulses should be input with the data in a 'don't care' state for the 12th clock pulse. See Figure 2.

When the chip first powers on, an internal power on reset signal is generated which resets the SR and latches to zero so that the display will be off.

## Timing Diagram

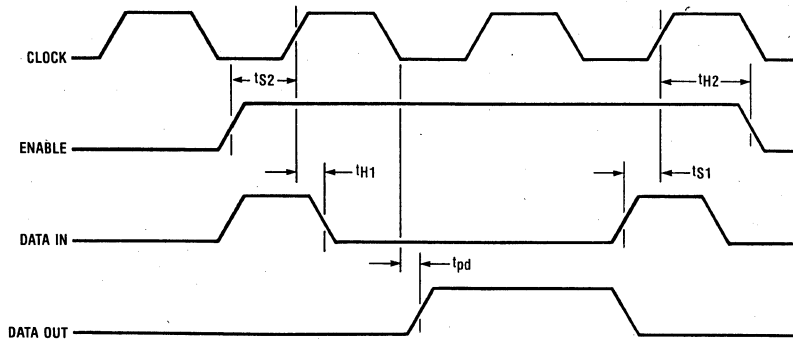


Figure 3.

# MM58201 Multiplexed LCD Driver

## General Description

The MM58201 is a monolithic CMOS LCD driver capable of driving up to 8 backplanes and 24 segments. A 192-bit RAM stores the data for the display. Serial input and output pins are provided to interface with a controller. An RC oscillator generates the timing necessary to refresh the display. The magnitude of the driving waveforms can be adjusted with the  $V_{TC}$  input to optimize display contrast. Four additional bits of RAM allow the user to program the number of backplanes being driven, and to designate the driver as either a master or slave for cascading purposes. When two or more drivers are cascaded, the master chip drives the backplane lines, and the master and each slave chip drive 24 segment lines. Synchronizing the cascaded drivers is accomplished by tying the RC OSC pins together and the BP1 pins together.

The MM58201 is packaged in a 40-lead dual-in-line package.

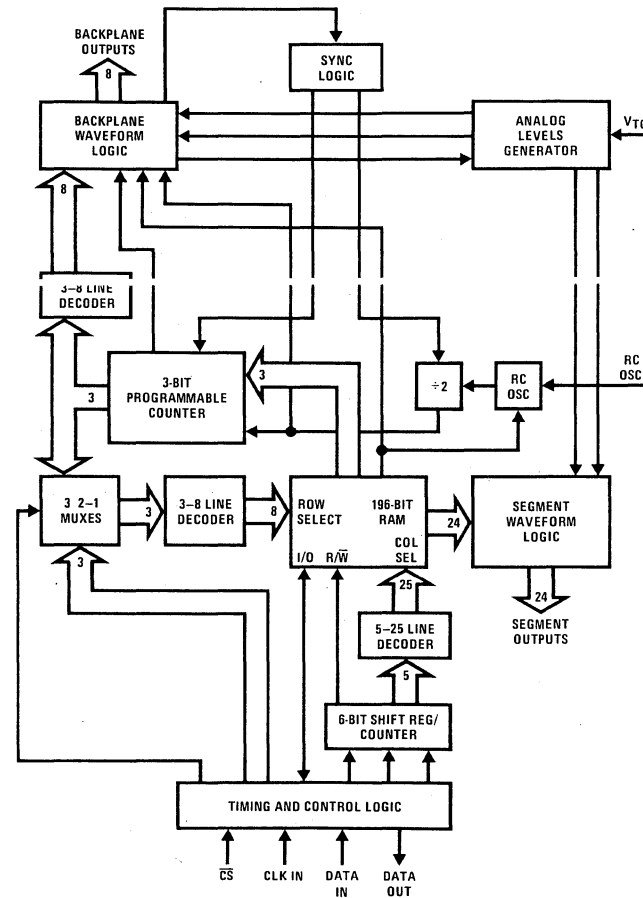
## Features

- Drives up to 8 backplanes and 24 segment lines
- Stores data for display
- Cascadable
- Low power
- Fully static operation

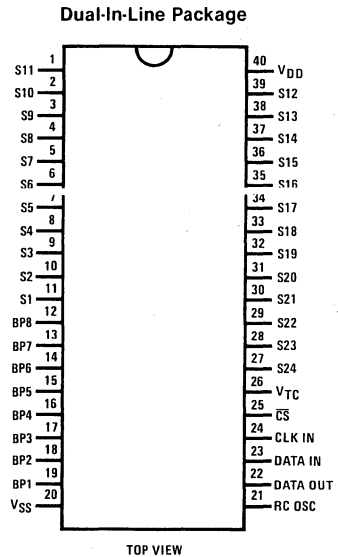
## Applications

- Dot matrix LCD driver
- Multiplexed 7-segment LCD driver
- Serial in/serial out memory

## Block Diagram


**FIGURE 1**

## Connection Diagram



Order Number MM58201N  
NS Package N24A

**FIGURE 2**



## Absolute Maximum Ratings

Voltage at Any Pin	$V_{SS} - 0.3V$ to $V_{SS} + 18V$
Operating Temperature Range	0°C to 70°C
Storage Temperature Range	-65°C to +150°C
Package Dissipation	500 mW
Operating $V_{DD}$ Range	$V_{SS} + 7.0V$ to $V_{SS} + 18.0V$
Lead Temperature (Soldering, 10 seconds)	300°C

## DC Electrical Characteristics

 Min/max limits apply across temperature range unless otherwise noted.

Parameter		Conditions	Min	Typ	Max	Units
$I_{CC}$	Quiescent Supply Current				0.3	mA
$V_{IN(1)}$	Logical "1" Input Voltage		0.45 $V_{DD}$		$V_{DD} + 0.3$	V
$V_{IN(0)}$	Logical "0" Input Voltage		$V_{SS} - 0.3$		1.0	V
$V_{OUT(0)}$	Logical "0" Output Voltage	$I_{SINK} = 0.6$ mA			0.4	V
$I_{OUT(1)}$	Logical "1" Output Leakage Current	$V_{OUT} = V_{DD}$	0		± 10	µA
$I_{IN(1)}$	Logical "1" Input Leakage Current	$V_{IN} = V_{DD}$	0		1.0	µA
$I_{IN(0)}$	Logical "0" Input Leakage Current	$V_{IN} = V_{SS}$	-1.0		0	µA
$V_{TC}$	Input Voltage		4.5		$V_{DD} + 0.3$	V
$V_{TC}$	Input Impedance		10		30	kΩ
$Z_{OUT}$	Output Impedance	Backplane and Segment Outputs			10	kΩ
	DC Offset Voltage	Between Any Backplane and Segment Output	0		± 10	mV

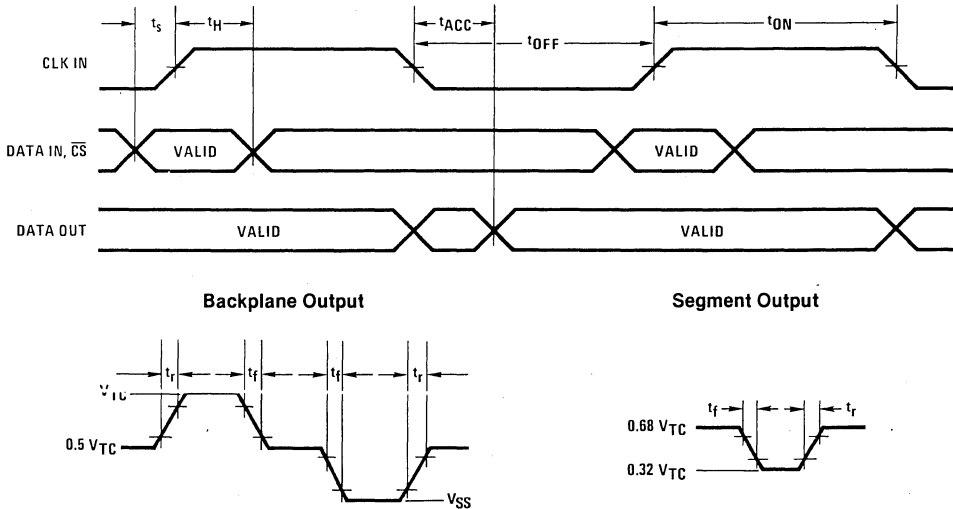
## AC Electrical Characteristics

 $T_A$  and  $V_{DD}$  within operating range unless otherwise noted.

Parameter		Conditions	Min	Typ	Max	Units
$f_{OSC}$	Oscillator Frequency*		128 $\eta$		400 $\eta$	Hz
$f_{CLK IN}$	Clock Frequency		DC		100	kHz
$t_{ON}$	Clock Pulse Width		5.0			µs
$t_{OFF}$	Clock OFF Time		5.0			µs
$t_s$	Input Data Set-Up Time		2.0			µs
$t_H$	Input Data Hold Time		1.0			µs
$t_{ACC}$	Access Time		5.0			µs
$t_r$	Rise Time	Backplane, Segment Outputs $C_L = 2000$ pF			60	µs
$t_f$	Fall Time	Backplane, Segment Outputs $C_L = 2000$ pF			60	µs

\*  $\eta$  is the number of backplanes programmed.

# Switching Time Waveforms



## Functional Description

A block diagram of the MM58201 LCD driver is shown in Figure 1. A connection diagram is shown in Figure 2.

### Serial Inputs and Output

A negative going edge on the  $\overline{CS}$  input initiates a frame. The  $\overline{CS}$  input must then stay low for at least one rising edge of CLK IN, and may not be pulsed low again for the next 31 clocks. At least one clock must occur while  $\overline{CS}$  is high. If CLK IN is held at a logic "1",  $\overline{CS}$  is disabled. This allows the signal that drives  $\overline{CS}$  to be used for other purposes when the MM58201 is not being addressed.

CLK IN latches data from the DATA IN input on its rising edge. Data from the DATA OUT pin changes on the falling edge of CLK IN and is valid before the next rising edge.

The first five bits of data following  $\overline{CS}$  are the address bits (Figure 3). The address selects the column where the operation is to start. Bit 1 is the MSB and bit 5 is the LSB. The sixth bit is the read/write bit. A logic "1" specifies a read operation and a logic "0" specifies a write operation. The next 24 bits are the data bits. The first data bit corresponds to the BP1 row of the display, the second data bit to the BP2 row, and so on. After the eighth and sixteenth data bits, the column pointer is incremented. When starting address 10110 or 10111 is specified, the column pointer increments from 10111 to 00000.

During a read or write cycle, the LCD segment outputs do not reflect the data in the RAM. To avoid disrupting the pattern viewed on the display, the read or write cycle time should be kept short. Since the LCD turn-on time can be as little as 30 ms, a clock rate of at least 10 kHz would be required in order to address the entire contents of the RAM

within that time interval. The formula below can be used to estimate the minimum clock rate:

$$f_{CLK IN} = (300 + 7 t_s) / t_{LCD}$$

where  $t_s$  is the processor's set-up time between each read or write cycle, and  $t_{LCD}$  is the minimum turn on or turn off time of the LCD as specified by the LCD manufacturer.

The DATA OUT output is an open drain N-channel device to  $V_{SS}$  (Figure 4). With an external pull-up this configuration allows the controller to operate at a lower supply voltage, and also permits the DATA OUT output to be wired in parallel with the DATA OUT outputs from any other drivers in the system.

To program the number of backplanes being driven and the  $M/\overline{S}$  bit, load address 11000, a write bit, three bits for the number of backplanes (Table I), and the  $M/\overline{S}$  bit. The remaining 20 data bits will be ignored but it is necessary to provide 21 more clocks before initiating another frame.

TABLE I. BACKPLANE SELECT

Number of Backplanes	B2	B1	B0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1

# Functional Description (Continued)

## RC OSC Pin

This oscillator generates the timing required for multiplexing the liquid crystal display. The oscillator operates at a frequency that is  $4\eta$  times the refresh rate of the display, where  $\eta$  is the number of backplanes programmed. Since the refresh rate should be in the range from 32 Hz to 100 Hz, the oscillator frequency must be:

$$128\eta \leq f_{OSC} \leq 400\eta$$

The frequency of oscillation is related to the external R and C components in the following way:

$$f_{OSC} = \frac{1}{1.25 RC} \pm 30\%$$

The value used for the external resistor should be in the range from 10 k $\Omega$  to 1 M $\Omega$ .

The value used for the external capacitor should be less than 0.005  $\mu$ F.

## V<sub>TC</sub> Pin

The V<sub>TC</sub> pin is an analog input that controls the contrast of the segments on the LCD. If eight backplanes are being driven ( $\eta = 8$ ), a voltage of typically 8V is required at 25°C. The voltage for optimum contrast will vary from display to display. It also has a significant negative temperature coefficient.

The voltage source on the V<sub>TC</sub> input must be of relatively low impedance since the input impedance of V<sub>TC</sub> ranges from 10 k $\Omega$  to 30 k $\Omega$ . A suitable circuit is shown in Figure 5.

In a standby mode, the V<sub>TC</sub> input can be set to V<sub>SS</sub>. This reduces the supply current to less than 300  $\mu$ A per driver.

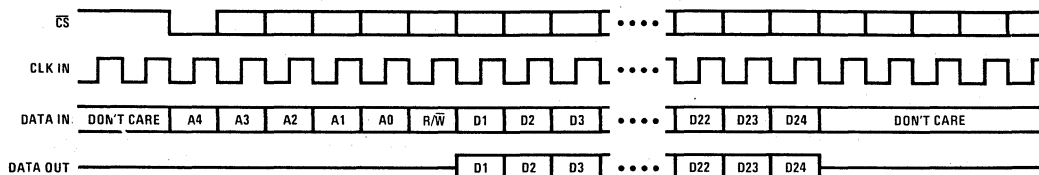
## Backplane and Segment Outputs

Connect the backplane and segment outputs directly to the LCD row and column lines. The outputs are designed to drive a display with a total ON capacitance of up to 2000 pF.

The output structure consists of transmission gates tapped off of a resistor string driven by V<sub>TC</sub> (Figure 6).

A critical factor in the lifetime of an LCD is the amount of DC offset between a backplane and segment signal. Typically, 50 mV of offset is acceptable. The MM58201 guarantees an offset of less than 10 mV.

The BP1 output is disabled when the M $\bar{S}$  bit is set to zero. This allows the BP1 output from the master chip to be connected directly to it so that synchronizing signals can be generated. Synchronization occurs once each refresh cycle, so the cascaded chips are assured of remaining synchronized.



	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23	S24		
BP1														D1	D9	D17										B2
BP2														D2	D10	D18										B1
BP3														D3	D11	D19										B0
BP4														D4	D12	D20										M/S
BP5														D5	D13	D21										
BP6														D6	D14	D22										
BP7														D7	D15	D23										
BP8														D8	D16	D24										
A4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
A3	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1
A2	0	0	0	0	1	1	1	1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0
A1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	1	1	0	0	1	1	0	0	1	1	0
A0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0

Diagram above shows where data will appear on display if starting address 01100 is specified in data format.

FIGURE 3. Data Format

# Functional Description (Continued)

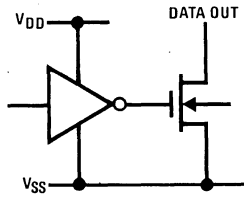


FIGURE 4. DATA OUT Structure

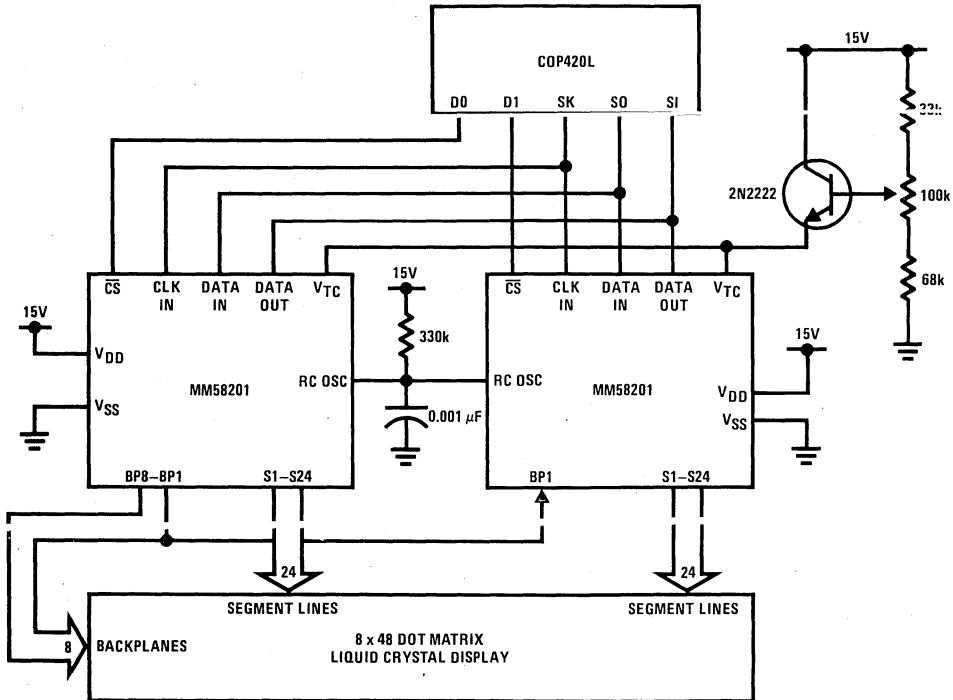


FIGURE 5. Typical Application

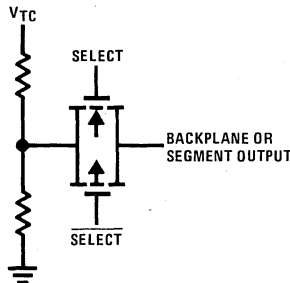


FIGURE 6. Structure of LCD Outputs

# MM58248, MM58241 High Voltage Display Drivers

## General Description

The MM58248 series are monolithic MOS integrated circuits utilizing a combined CMOS/Bipolar process with both MOS and Junction F.E.T. devices. They are available in 40-pin dual-in-line packages, or as dice. Each output can source 1mA at 2V maximum output voltage, and also has an internal Junction F.E.T. to the display supply voltage which can be up to 60V. The possibility of brightness control is also provided.

- MICROWIRE™ compatible (MM58241)
- Simple to cascade (MM58241)
- Wide supply operation
- TTL compatible inputs
- Software compatible with NS display driver family
- Compatible with VF, high voltage LCD, and colloidal displays

## Features

- Direct interface to 60V VF display
- Brightness and display blanking control input (MM58241)
- No resistors needed
- No load signal required (MM58248)

## Applications

- COPS™ or microprocessor displays
- Instrumentation readouts
- Integrated dashboard displays
- Word processor text display

COPS AND MICROWIRE are trademarks of National Semiconductor Corp.

## Block Diagram

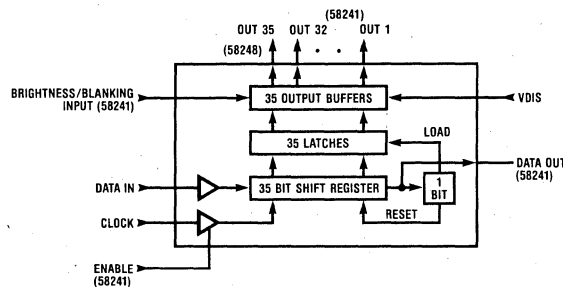
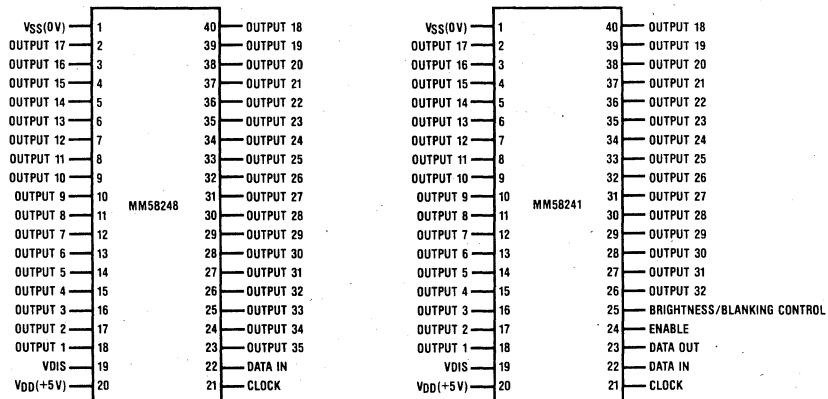


Figure 1. Block Diagram

## Connection Diagrams



Order Number MM58248N, MM58241N  
NS Package Number N40A

Figure 2.

## Absolute Maximum Ratings

Voltage at Any Input Pin	$V_{DD} + 0.3V$ to $V_{SS} - 0.3V$
Voltage at Any Display Pin	$V_{DD}$ to $V_{DD} - 65V$
Operating Temperature	$-40^{\circ}C$ to $85^{\circ}C$
Storage Temperature	$-65^{\circ}C$ to $150^{\circ}C$
Power Dissipation	500mW at $85^{\circ}C$ 750mW at $25^{\circ}C$
Junction Temperature	$130^{\circ}C$
Lead Temperature (Soldering, 10 seconds)	$300^{\circ}C$

## Electrical Characteristics

$T_A$  within operating range,  $V_{DD} = 5V \pm 0.5V$ ,  $V_{SS} = 0V$ , unless otherwise specified

Parameter	Conditions	Min.	Typ.	Max.	Units
Power Supply					
$V_{DD}$	$V_{SS} = 0V$	4.5	5.0	5.5	V
$V_{DIS}$	$V_{DD} = 5V$ $V_{SS} = 0V$	-10		-55	V
Power Supply Current					
$I_{SS}$	$V_{DD} = 5V$ $V_{SS} = 0V$		10	100	$\mu A$
$I_{DIS}$	$V_{DIS} = -55V$		5	12	mA
Input Logic Level					
Data In, Clock Enable	$V_{DD} = 5.0 \pm 0.5V$ $V_{SS} = 0$				
Logic "0"		$V_{SS}$		0.8	V
Logic "1"		2.4		$V_{DD}$	
Input Current					
Data In, Clock Enable				10	$\mu A$
Output Impedance					
Output Off	$V_{DIS} = -40V$ , $V_{OUT} = V_{DIS} + 2V$		200		k $\Omega$
Output On	$I_{SOURCE} = 1mA$			2	k $\Omega$
Input Clock					
Frequency	$V_{DD} = 4.5V$			500	kHz
Rise Time				200	ns

## Functional Description

This series of products is specifically designed to drive either 4 or 5 digit non-multiplexed high voltage displays (e.g., dynamic scattering LCD or gas discharge) or multi-digit dot matrix high voltage displays (e.g., VF). Character generation is done externally in the microprocessor, with a serial data path to the display driver. Two data transfer modes and display brightness controls exist. The MM58248 uses two signals, data and clock, with a format of a leading '1' followed by the 35 data bits, hence allowing data transfer without an additional load signal. Display brightness can be achieved through software control with the MM58248. The MM58241 uses a standard MICROWIRE™ interface for data transfer. Display brightness is determined by the duty cycle of the brightness/blinking input. Full brightness is obtained with a logic '0' at this input and blanking with a logic '1'. A block diagram is shown in Figure 1.

Figure 2 shows the pinout of the MM58248 series. Bit 1 is the first bit to be loaded (following the start bit of MM58248). A logic '1' at the input will turn on the appropriate display segment output. Figure 5 describes the combined MOS and Junction F.E.T. output structure. The Junction F.E.T. has a pinch-off voltage in excess of 60V and may be viewed simply as a high impedance resistor.

MICROWIRE is a trademark of National Semiconductor Corp.

Figure 4 illustrates both possible microprocessor interfaces. In 4a, a start bit of logic '1' precedes the 35 bits of data. At the 36th clock a LOAD signal is generated synchronously with the high state of the clock, which loads the 35 bits of shift registers into the latches. At the low state of the clock a RESET signal is generated which clears all the shift registers for the next set of data. Hence a complete set of 36 clocks is needed or the shift register will not clear.

In Figure 4b, the ENABLE signal acts as an envelope and only while this signal is at a logic '1' does the circuit accept CLOCK input signals. Data is transferred and shifted in the internal shift register on the rising clock edge, i.e., '0' - '1' transition. When the ENABLE signal goes low, the contents of the shift register are latched and the display will show new data. During data transfer, the display will continue to show old data. DATA OUT is also provided in this mode, being output on the falling clock edge.

When the chip first powers on, an internal reset is generated which resets all registers and latches. The chip returns to normal operation on application of the start bit and the first clock for MM58248 or an application of ENABLE for MM58241. All interface signals from the microprocessor should be inactive at power on.

## Timing Diagram

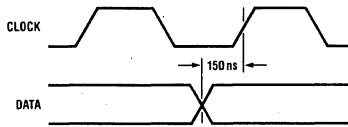


Figure 3.

## Data Format

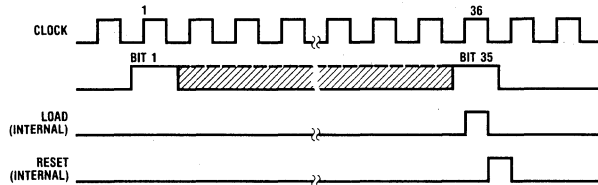


Figure 4a. MM58248 Microprocessor Interface

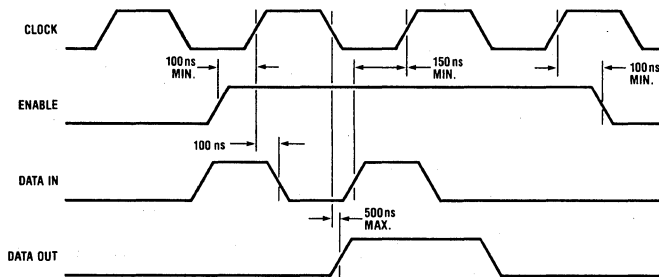
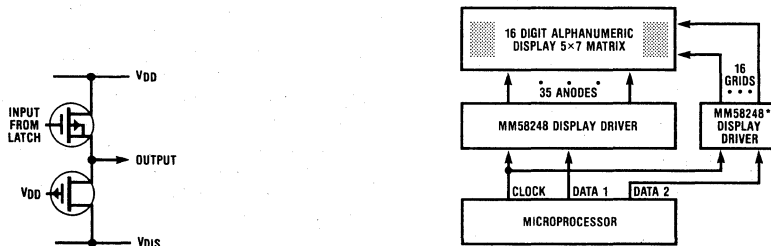


Figure 4b. MM58241 Microprocessor Interface

## Typical Application



\*For high current displays, MM58348 outputs may need to be paralleled or, as an alternative, the DS8881 may be required to be used as a grid driver.

Figure 5. Output Structure

Figure 6. Word Processor Application

# MM58348, MM58341 High Voltage Display Drivers

## General Description

The MM58348 series are monolithic MOS integrated circuits utilizing a combined CMOS/Bipolar process with both MOS and Junction F.E.T. devices. They are available in 40-pin molded dual-in-line packages or as dice. Each output can source 3mA at 1V maximum output voltage, and also has an internal Junction F.E.T. to the display supply voltage which can be up to 32V. The possibility of brightness control is also provided.

## Features

- Direct interface to 32V VF display
- Brightness and display blanking control input (MM58341)
- No resistors needed
- No load signal required (MM58348)

- MICROWIRE™ compatible (MM58341)
- Simple to cascade (MM58341)
- Wide supply operation
- TTL compatible inputs
- Software compatible with NS display driver family
- Compatible with VF, high voltage LCD, and colloidal displays

## Applications

- COPST™ or microprocessor displays
- Instrumentation readouts
- Integrated dashboard displays
- Word processor text display

COPS and MICROWIRE are trademarks of National Semiconductor Corp.

## Block Diagram

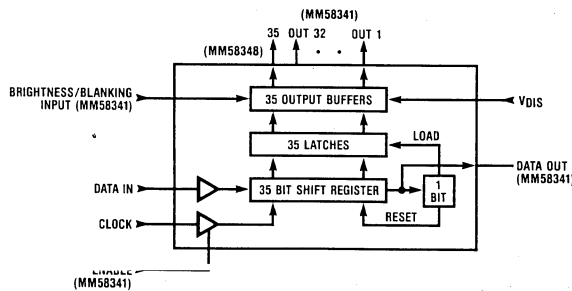
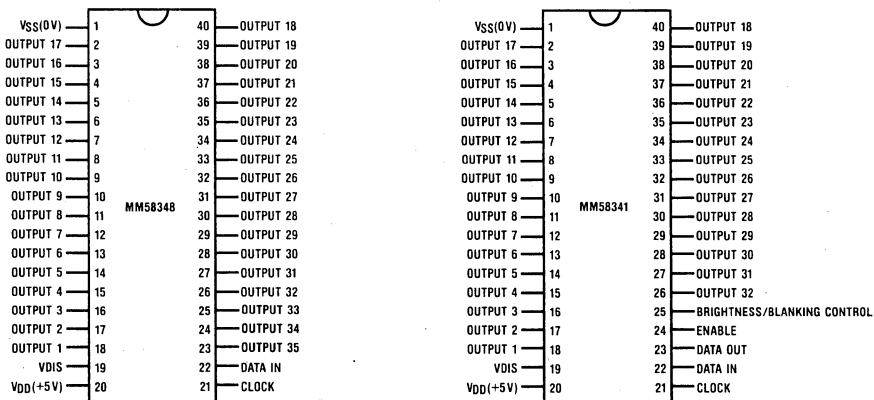


Figure 1. Block Diagram

## Connection Diagrams



Order Number MM58348, MM58341  
NS Package N40A

Figure 2.



## Absolute Maximum Ratings

Voltage at Any Input Pin	$V_{DD} + 0.3V$ to $V_{SS} - 0.3V$
Voltage at Any Display Pin	$V_{DD}$ to $V_{DD} - 40V$
Operating Temperature	$-40^{\circ}C$ to $85^{\circ}C$
Storage Temperature	$-65^{\circ}C$ to $150^{\circ}C$
Power Dissipation	500mW at $85^{\circ}C$ 750mW at $25^{\circ}C$
Junction Temperature	$130^{\circ}C$
Lead Temperature (Soldering, 10 seconds)	$300^{\circ}C$

## Electrical Characteristics

$T_A$  within operating range,  $V_{DD} = 5V \pm 0.5V$ ,  $V_{SS} = 0V$ , unless otherwise specified

Parameter	Conditions	Min.	Typ.	Max.	Units
Power Supply $V_{DD}$ $V_{DIS}$	$V_{SS} = 0V$ $V_{DD} = 5V$ $V_{SS} = 0V$	4.5 -10	5.0	5.5 -27	V V
Power Supply Current $I_{SS}$ $I_{DIS}$	$V_{DD} = 5V$ $V_{SS} = 0V$ $V_{DIS} = -25V$		10 5	100 12	$\mu A$ mA
Input Logic Level Data In, Clock Enable  Logic "0" Logic "1"	$V_{DD} = 5.0 \pm 0.5V$ $V_{SS} = 0$		$V_{SS}$ 2.4	0.8 $V_{DD}$	V
Input Current Data In, Clock Enable				10	$\mu A$
Output Impedance Output Off Output On	$V_{DIS} = -27V$ , $V_{OUT} = V_{DIS} + 2$ $I_{SOURCE} = 3mA$		200 250	400	k $\Omega$ $\Omega$
Input Clock Frequency Rise Time	$V_{DD} = 4.5V$			500 200	kHz ns

## Functional Description

This series of products is specifically designed to drive either 4 or 5 digit non-multiplexed high voltage displays (e.g., dynamic scattering LCD or gas discharge) or multi-digit dot matrix high voltage displays (e.g., VF). Character generation is done externally in the microprocessor, with a serial data path to the display driver. Two data transfer modes and display brightness controls exist. The MM58348 uses two signals, data and clock, with a format of a leading '1' followed by the 35 data bits, hence allowing data transfer without an additional load signal. Display brightness can be achieved through software control with the MM58348. The MM58341 uses a standard MICROWIRE™ interface for data transfer. Display brightness is determined by the duty cycle of the brightness/blinking input. Full brightness is obtained with a logic '0' at this input and blanking with a logic '1'. A block diagram is shown in Figure 1.

Figure 2 shows the pinout of the MM58348 series. Bit 1 is the first bit to be loaded (following the start bit of MM58348). A logic '1' at the input will turn on the appropriate display segment output. Figure 5 describes the combined MOS and Junction F.E.T. output structure. The Junction F.E.T. has a pinch-off voltage in excess of 32V and may be viewed simply as a high impedance resistor.

Figure 4 illustrates both possible microprocessor interfaces. In 4a, a start bit of logic '1' precedes the 35 bits of data. At the 36th clock a LOAD signal is generated synchronously with the high state of the clock, which loads the 35 bits of shift registers into the latches. At the low state of the clock a RESET signal is generated which clears all the shift registers for the next set of data. Hence a complete set of 36 clocks is needed or the shift register will not clear.

In Figure 4b, the ENABLE signal acts as an envelope and only while this signal is at a logic '1' does the circuit accept CLOCK input signals. Data is transferred and shifted in the internal shift register on the rising clock edge, i.e., '0' - '1' transition. When the ENABLE signal goes low, the contents of the shift register are latched and the display will show new data. During data transfer, the display will continue to show old data. DATA OUT is also provided in this mode, being output on the falling clock edge.

When the chip first powers on, an internal reset is generated which resets all registers and latches. The chip returns to normal operation on application of the start bit and the first clock for MM58348 or an application of ENABLE for MM58341. All interface signals from the microprocessor should be inactive at power on.

## Timing Diagram

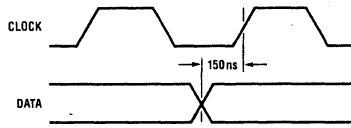


Figure 3.

## Data Format

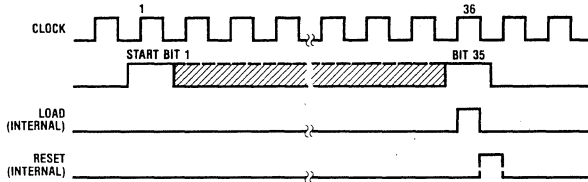


Figure 4a. MM58348 Microprocessor Interface

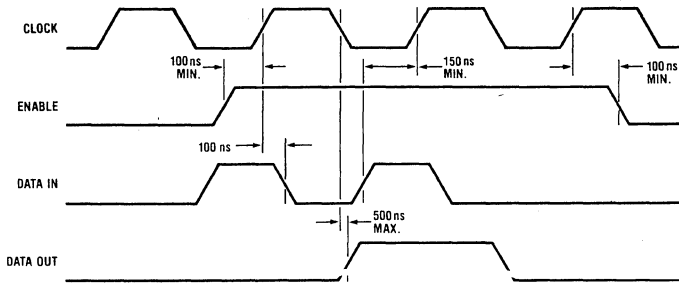


Figure 4b. MM58341 Microprocessor Interface

## Typical Application

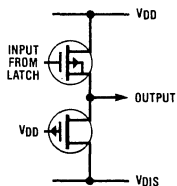
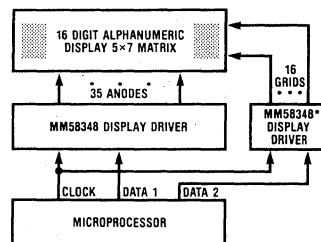
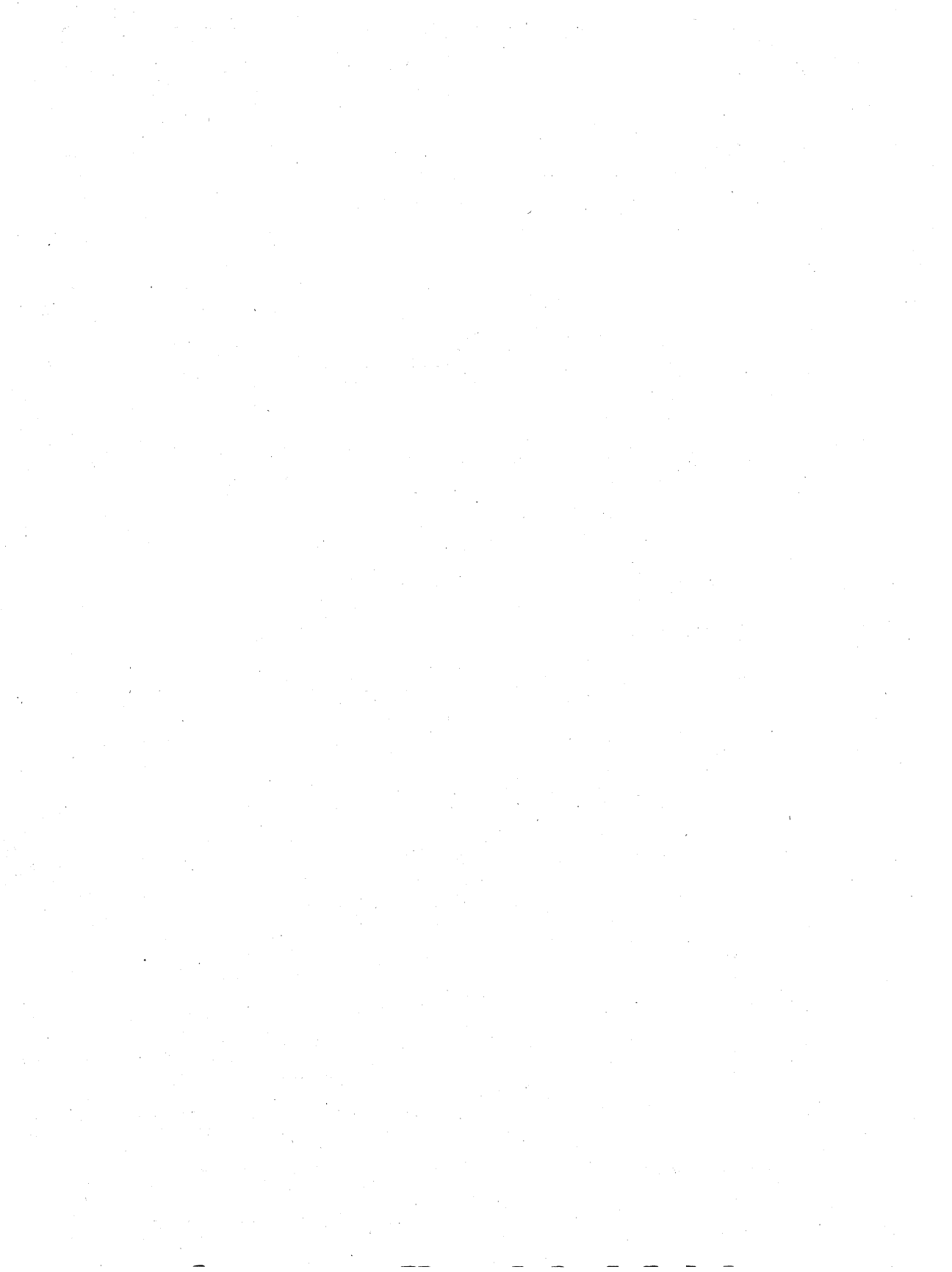


Figure 5. Output Structure



\*For high current displays, MM58348 outputs may need to be paralleled or, as an alternative, the DS8881 may be required to be used as a grid driver.

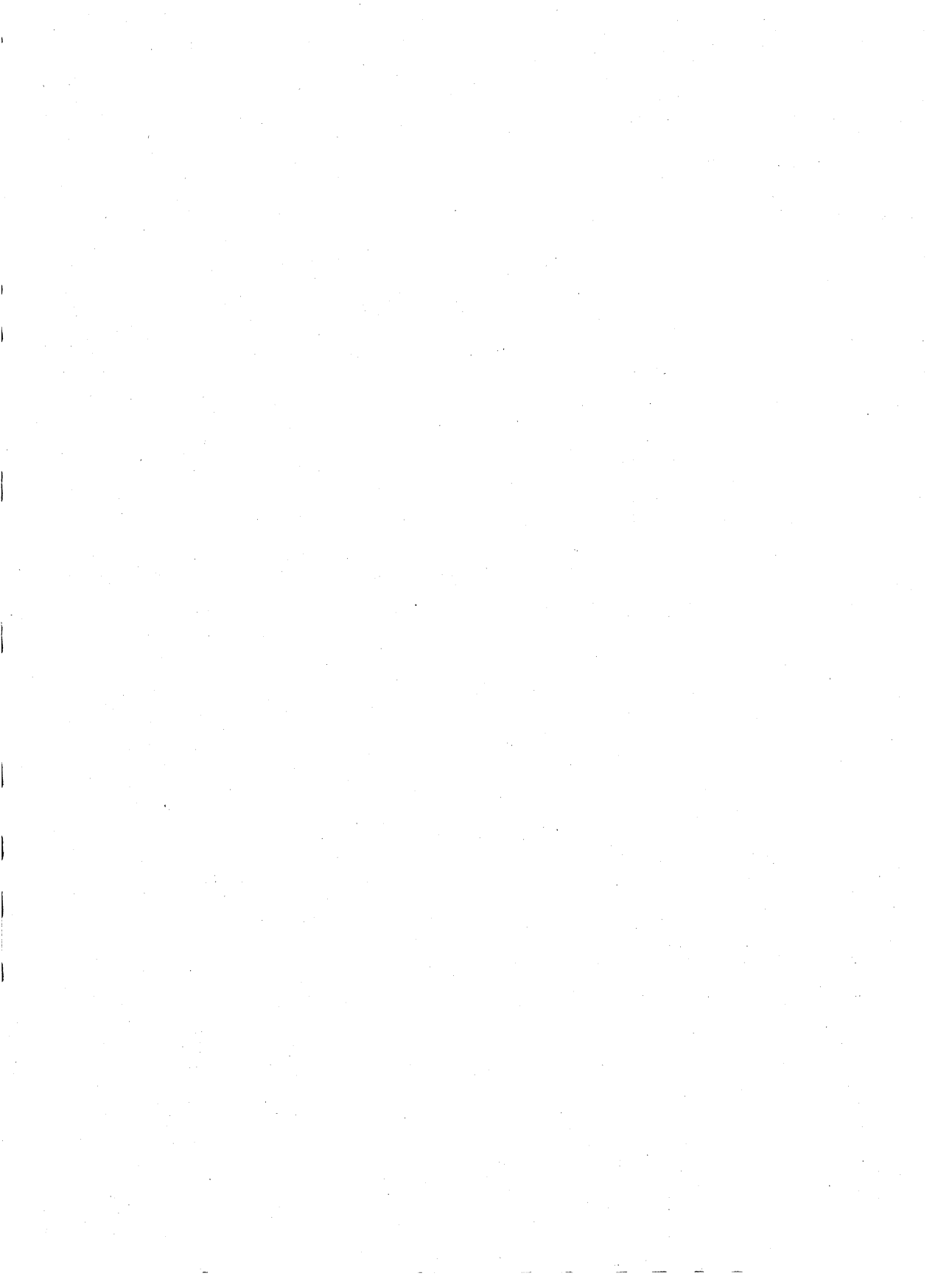
Figure 6. Word Processor Application





**Section 6**  
**Standard**  
**Controllers**





# MM57409 Super Number Cruncher

## General Description

The MM57409 Super Number Cruncher is designed to function as a peripheral arithmetic processor in microprocessor applications. Data and instructions are transferred asynchronously between processor and peripheral using the standard 8-bit MICROBUS™. Software development is greatly simplified when using the MM57409's calculator keyboard level language. This means that complex arithmetic functions can be incorporated in microprocessor software quickly and easily by any programmer familiar with the operation of a scientific calculator.

The MM57409 is also capable of stand alone operation. Besides arithmetic operations, the device has internal number storage, input/output instructions and test and branch capability. In the stand-alone mode, an 8-bit address is present on the PC<sub>0</sub>-PC<sub>7</sub> pins for interface to an external program PROM, ROM, or RAM.

## Features

- Scientific calculator instructions (RPN)
  - Up to 12-digit mantissa, 2-digit exponent
  - Four-register stack, one memory register
  - Trigonometric functions, logarithmic functions, Y<sub>x</sub>, e<sup>x</sup>, pi
  - Error flag generation and recovery

- Flexible input/output
  - Multidigit I/O instructions (IN, OUT) with floating point or scientific notations
  - Programmable mantissa digit count for IN, OUT instructions
  - Sense input and flag outputs
- Branch control
  - Conditional and unconditional program branching
- Interface simplicity
  - On-chip clock OSC
  - Generates all I/O control signals
  - MICROBUS interface

## Applications

- Instruments
- Microprocessor/minicomputer peripheral
- Test equipment
- Process controllers

MICROBUS is a trademark of National Semiconductor Corp.

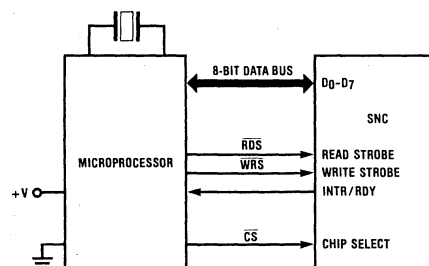
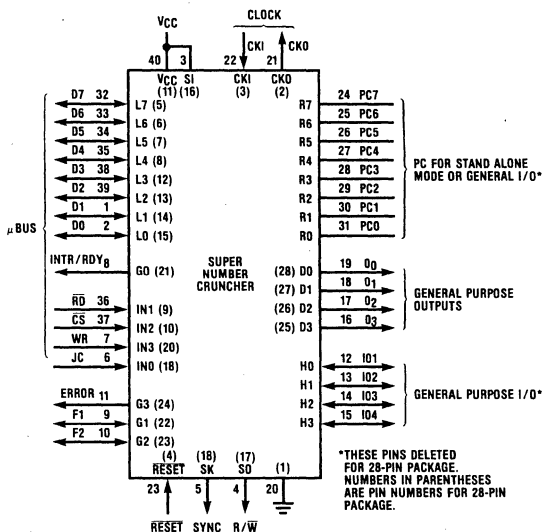


Figure 1. Super Number Cruncher — Pinout

Super Number Cruncher Interface with 8-Bit Microprocessor

**Data Entry Instructions**

0	Mantissa or exponent digits. On first digit (d),
1	if prior code was not EN (Enter), get stack push: was
2	$Z \rightarrow t$
3	$Y \rightarrow Z$
4	$X \rightarrow Y$
5	$d \rightarrow X$
6	If prior code was EN, get simply d.
7	Set number entry mode. See number entry description.
8	
9	
DP	Decimal point. Digits that follow will be mantissa fraction. If first "numeric" entry, initiates number entry mode as above.
EE	Enter Exponent. Digits that follow will be exponent. If first "numeric" entry, initiates number entry mode as above and loads 1 to mantissa.
CS	Change Sign. If EE instruction was executed after last number entry initiation, changes exponent sign X; else changes sign X mantissa. Does not initiate number entry.
Pi	3.14159265359 $\rightarrow X$ ; if first numeric entry, initiate number entry mode (stack push) as above.
AIN1	Single-Digit Asynchronous input initiates number entry as above. See input/output description.
NOP1	No operation. Do nothing. Status not altered in any way.

**Data Input**

IN	Multidigit input instruction — SNC accepts all required data for input. See input/output description for further explanation.
AIN2	Asynchronous input 2. 2-byte instruction. Write a single digit, any digit, in x. Second byte of form $Nx$ where $N=0-F$ for digit address in register $x = \text{BCD data}$ . See input/output description for further explanation.
IDPC	Load PC/8-bit general I/O port with data contained in next byte. 2-byte instruction.
NOP2	Terminate number entry; no other operation.

**Mode and Flag Instructions**

RAD	Set radian angular mode.
DEG	Set degrees angular mode default mode.
RIO	Enable R as general I/O.
RPC	Enable R as program counter.
NORND	Disable round to MDC on output.
RND	Disable round to MDC on output default mode.
FLP	Set floating point I/O mode.
SCI	Set scientific notation I/O mode—default mode.
SIF1	Set internal flag 1.
RIF1	Reset internal flag 1.
SIF2	Set internal flag 2.
RIF2	Reset internal flag 2.

SIF3	Set internal flag 3.
RIF3	Reset internal flag 3.
SIF4	Set internal flag 4.
RIF4	Reset internal flag 4.

**Math Instructions**

CLR <sub>X</sub>	$0 \rightarrow x$
EN	Enter, terminate number entry and push stack. $z \rightarrow t$ $y \rightarrow z$ $x \rightarrow y$ same number in x and y.
NOP2	Terminate number entry, no other operation.
ROLL	Roll Stack $\rightarrow x \rightarrow y \rightarrow z \rightarrow t \rightarrow \rightarrow$
SIN	$\text{Sin}(x) \rightarrow x; y, z, t, m$ unchanged.
COS	$\text{COS}(x) \rightarrow x; y, z, t, m$ unchanged.
TAN	$\text{Tan}(x) \rightarrow x; y, z, t, m$ unchanged.
ARCSIN	$\text{Sin}^{-1}(x) \rightarrow x; y, z, t, m$ unchanged.
ARCCOS	$\text{COS}^{-1}(x) \rightarrow x; y, z, t, m$ unchanged.
ARCTAN	$\text{Tan}^{-1}(x) \rightarrow x; y, z, t, m$ unchanged.
NOP2	Terminates number entry, no other operation.
ECLR	Clear Error Flag.
RTD	Convert x; radians to degrees $y, z, t, m$ unchanged.
DTR	Convert x; degrees to radians $y, z, t, m$ unchanged.
POP	Pop Stack: $y \rightarrow x$ $z \rightarrow y$ $t \rightarrow z$ $0 \rightarrow t$
MCLR	Clear all internal registers and outputs; 10 MDC scientific notation; round to MDC on output.
KEY	Exchange $x, y$ $x \leftrightarrow y$
EX	$e^x \rightarrow x; y, z, t, m$ unchanged.
10 <sup>X</sup>	$10^x \rightarrow x; y, z, t, m$ unchanged.
SQ	$x^2 \rightarrow x; y, z, t, m$ unchanged.
SQRT	$(x)^{0.5} \rightarrow x; y, z, t, m$ unchanged.
LN	$\ln x \rightarrow x; y, z, t, m$ unchanged.
LOG	$\log x \rightarrow x; y, z, t, m$ unchanged.
1/X	$1/x \rightarrow x; y, z, t, m$ unchanged.
YX	$y^x \rightarrow x; z \rightarrow y, t \rightarrow z, 0 \rightarrow t$ .
+	$x+y \rightarrow x; z \rightarrow y, t \rightarrow z, 0 \rightarrow t$ .
-	$x-y \rightarrow x; z \rightarrow y, t \rightarrow z, 0 \rightarrow t$ .
x	$x \cdot y \rightarrow x; z \rightarrow y, t \rightarrow z, 0 \rightarrow t$ .
/	$x/y \rightarrow x; z \rightarrow y, t \rightarrow z, 0 \rightarrow t$ .
NOP2	Terminate number entry, no other operation.
LSH	Left shift x mantissa, DP unchanged, MSD saved in guard/link digit.
RSH	Right shift x mantissa, DP unchanged, link/guard digit MSD.

**Test Instructions**

TJC	If jump condition (input JC) true, load PC with data in second byte.
TX=0	If X=0, load PC with data in second byte.
TX<TO	If X<0, load PC with data in second byte.
TXF	If $1 \times 1 < 0$ , load PC with data in second byte.
TERR	If error flag set, load PC with data in second byte.
JMP	Load PC with data in second byte.
TMNZ	If M=0, load PC with data in second byte.
TM=0	If M=0, load PC with data in second byte.
TF1	If F1=1, load PC with data in second byte.
IBMNZ	Increment M matissa. If M=0, load PC with data in second byte.
DBMNZ	Decrement M matissa. If M=0, load PC with data in second byte.
TF2	If F2=1, load PC with data in second byte.
TIF1	If internal flag 1=1, load PC with data in second byte.
TIF2	If internal flag 2=1, load PC with data in second byte.
TIF3	If internal flag 3=1, load PC with data in second byte.
TIF4	If internal flag 4=1, load PC with data in second byte.

**Memory Instructions**

XEM	$x \leftrightarrow M$ ; exchange x, memory.
MS	$x \rightarrow M$ ; store x in memory.
MR	$M \rightarrow x$ ; stack pushed; $M \rightarrow x \rightarrow y \rightarrow z \rightarrow t$
M+	$M+x \rightarrow M$ ; x,y,z,t unchanged.
M-	$M-x \rightarrow M$ ; x,y,z,t unchanged.
M^	$M \wedge x \rightarrow M$ ; x,y,z,t unchanged.
M/	$M/x \rightarrow M$ ; x,y,z,t unchanged.
CLRM	$0 \rightarrow M$ .
NOP2	Terminate Number Entry, no other operation.

**Digit Count Control**

SMDC1	Set Mantissa Digit Count = 1.
SMDC2	Set Mantissa Digit Count = 2.
SMDC3	Set Mantissa Digit Count = 3.
SMDC4	Set Mantissa Digit Count = 4.
SMDC5	Set Mantissa Digit Count = 5.
SMDC6	Set Mantissa Digit Count = 6.
SMDC7	Set Mantissa Digit Count = 7.
SMDC8	Set Mantissa Digit Count = 8.
SMDC9	Set Mantissa Digit Count = 9.
SMDC10	Set Mantissa Digit Count = 10.
SMDC11	Set Mantissa Digit Count = 11.
SMDC12	Set Mantissa Digit Count = 12.
NOP2	
NOP2	Terminate number entry,
NOP2	no other operation.
NOP2	

**Output Control Instruction**

ROFF	Tristate R port, disallowed if R is program counter.
RON	Enable R drives.
NOP2	Terminate number entry, no other operation.
SF1	Set F1 to 1.
PF1	F1 is pulsed high. If F1 is set, results in F1 being reset.
SF2	Set F2 to 1.
PF2	F2 is pulsed high. If F2 is set, results in F2 being reset.
NOP2	Terminate number entry; no other operation.
NOP2	
PRW	Active low pulse (low going) generated at R/W.
PRW	Active low pulse (low going) generated at R/W.
PRW	Active low pulse (low going) generated at R/W.



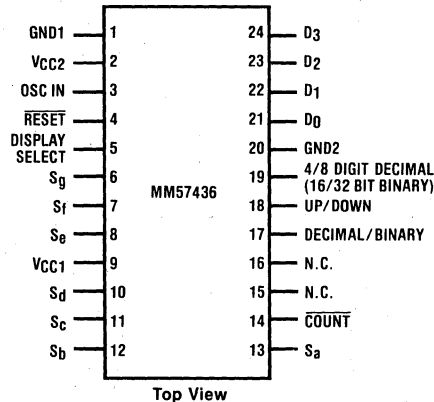
## MM57436 Decimal/Binary Up/Down Counter

### General Description

The MM57436 Counter, an NMOS silicon gate technology device, is designed to be a minimal solution Decimal/Binary Up/Down counter with display capability. The counter length is user selectable at 4 digits decimal (16 bits binary) or 8 digits decimal (32 bits binary). The device has the capability of direct drive of a 4 digit multiplexed LED display. In the 8-digit (32-bit) mode, the user may direct either the top four digits or lower four digits to the display. The MM57436 will run off an internal RC oscillator or the user may supply an external oscillator for greater precision in the count rate.

### Features

- Decimal or binary count
- Up or down count
- 4 or 8 digit (16 or 32 bit) counter length
- 4 digit, seven segment multiplexed LED display drive
- User display control
- Single supply operation
- Wide supply range (4.5V-9.5V)
- TTL compatible on inputs



Order Number MM57436N  
NS Package N24A

Pin	Description
OSC IN	Oscillator Input — External Oscillator or RC
Display Select	Control line to display upper or lower 4 digits (16 bits) of 8-digit (32-bit) counter
$S_A$ - $S_G$	Multiplexed 7-segment outputs
COUNT	Input for signal to be counted
Decimal/Binary	Counter mode control
Up/Down	Up-down count control
4/8 Digit (16/32 Bit Binary)	Counter length control
$D_0$ - $D_3$	Display digit strobes
$V_{CC1}$ , $V_{CC2}$	Power supply
GND1, GND2	Ground

Figure 1. Connection Diagram

## Absolute Maximum Ratings

Voltage at Any Pin Relative to GND <sub>1</sub>	-0.3V to +10V
Ambient Operating Temperature	0°C to +70°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 Seconds)	300°C
Power Dissipation	0.75 Watt at 25°C 0.4 Watt at 70°C

"Absolute Maximum Ratings" indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.

## DC Electrical Characteristics

0°C ≤ T<sub>A</sub> ≤ 70°C, 4.5V ≤ V<sub>CC</sub> ≤ 9.5V, unless otherwise specified

Parameter	Conditions	Min.	Typ.	Max.	Units
Operating Voltage (V <sub>CC</sub> )		4.5		9.5	V
Operating Supply Current	(all inputs and outputs open)			6.0	mA
Input Voltage Levels					
OSC IN, RESET Levels					
Logic High (V <sub>IH</sub> )		0.7V <sub>CC</sub>			V
Logic Low (V <sub>IL</sub> )				0.6	V
All Other Inputs					
Logic High (V <sub>IH</sub> )	V <sub>CC</sub> = 9.5V	3.0			V
Logic High (V <sub>IH</sub> )	V <sub>CC</sub> = 5V ± 10%	2.0			V
Logic Low (V <sub>IL</sub> )				0.8	V
Output Current Levels					
Output Sink Current					
D <sub>0</sub> -D <sub>3</sub> (I <sub>OL</sub> )	V <sub>CC</sub> = 9.5V, V <sub>OL</sub> = 1.0V	30			mA
	V <sub>CC</sub> = 4.5V, V <sub>OL</sub> = 1.0V	15			mA
S <sub>A</sub> -S <sub>G</sub> (I <sub>OL</sub> )	V <sub>CC</sub> = 9.5V, V <sub>OL</sub> = 0.4V	0.8			mA
	V <sub>CC</sub> = 4.5V, V <sub>OL</sub> = 0.4V	0.4			mA
Output Source Current					
S <sub>0</sub> -S <sub>6</sub> (I <sub>OH</sub> )	V <sub>CC</sub> = 9.5V, V <sub>OH</sub> = 2.0V	-3.0		-35	mA
	V <sub>CC</sub> = 6.0V, V <sub>OH</sub> = 2.0V	-3.0		∞	mA

## AC Electrical Characteristics

0°C ≤ T<sub>A</sub> ≤ 70°C, 4.5V ≤ V<sub>CC</sub> ≤ 9.5V, unless otherwise specified

Parameter	Conditions	Min.	Typ.	Max.	Units
OSC IN					
Frequency		100		266.67	kHz
Duty Cycle		40		60	%
Rise Time				1	μs
Fall Time				1	μs
Internal Time Base (= 4/Frequency)		15		40	μs
OSC IN Using RC					
Frequency	R = 56 kΩ ± 5%, C = 100 pF ± 10%	140		266.67	kHz
Internal Time Base (= 4/Frequency)		15		28	μs
Inputs					
Up/Down, Display Select					
t <sub>SETUP</sub>				8	μs
t <sub>HOLD</sub>				1	μs
Count					
t <sub>SETUP</sub>				2	μs
t <sub>HOLD</sub>				1	μs

# AC Electrical Characteristics (continued) $0^{\circ}\text{C} \leq T_A \leq 70^{\circ}\text{C}$ , $4.5\text{V} \leq V_{CC} \leq 9.5\text{V}$ , unless otherwise specified

Parameter	Conditions	Min.	Typ.	Max.	Units
Count Input Frequency	4 Digit Decimal Up Count OSC IN = 266.67 kHz OSC IN = 100 kHz			14.4	kHz
				5.43	kHz
	4 Digit Decimal Down Count OSC IN = 266.67 kHz OSC IN = 100 kHz			13.6	kHz
				5.13	kHz
	8 Digit Decimal Up Count OSC IN = 266.67 kHz OSC IN = 100 kHz			9.52	kHz
				3.57	kHz
	8 Digit Decimal Down Count OSC IN = 266.67 kHz OSC IN = 100 kHz			9.17	kHz
				3.44	kHz
	16 Bit Binary Up Count OSC IN = 266.67 kHz OSC IN = 100 kHz			16.3	kHz
				6.14	kHz
	16 Bit Binary Down Count OSC IN = 266.67 kHz OSC IN = 100 kHz			15.3	kHz
				5.76	kHz
	32 Bit Binary Up Count OSC IN = 266.67 kHz OSC IN = 100 kHz			11.2	kHz
				4.21	kHz
32 Bit Binary Down Count OSC IN = 266.67 kHz OSC IN = 100 kHz	10.3	kHz			
	3.86	kHz			
Pulse Width (= 8/OSC IN Frequency)	OSC IN = 100kHz	80			$\mu\text{s}$
	OSC IN = 266.67 kHz	30			$\mu\text{s}$
RESET Input Pulse Width	Resetting device while device running OSC IN = 100 kHz OSC IN = 266.67 kHz	160			$\mu\text{s}$
		60			$\mu\text{s}$

## Functional Description

The MM57436 will count pulses at its count input and will display 4 digits of the resultant count. Under user control the device will count in either decimal or binary and will either count up or count down. The user may also select which group of 4 digits (16 bits) is to be displayed.

The display is standard, seven-segment for the decimal counter. In the binary mode, hex characters are displayed as follows:

0-9, A, b, C, d, E, F

The mode controls of the MM57436 are as follows:

**Decimal/Binary** — With this pin left open or tied to  $V_{CC}$ , the MM57436 is a decimal counter. Connecting this pin to output D1 converts the MM57436 to a binary counter. This mode is a strap option and may *not* be changed while the device is running.

**4/8-Digit Decimal (16/32-Bit Binary)** — With this pin left open or tied to  $V_{CC}$  the MM57436 is a 4-digit decimal or 16-bit binary counter. Connecting this pin to ground converts the MM57436 to an 8-digit decimal or 32 bit binary counter. The counter length is a strap option and may *not* be changed while the device is running.

**Up/Down** — With this pin left open or at a logic "1" (positive logic) the MM57436 will increment its internal counter by 1 with every pulse input at the COUNT input. With this pin connected to ground or to a logic "0" (positive logic), the MM57436 will *decrement* its internal counter by 1 with every pulse at the COUNT input. This input may be tied high or low, may come from a switch or may be controlled by a logic signal. It may be changed by the user at any time. Note, if this input is to be controlled by a mechanical switch some external debounce protection may be required depending on the application. There is no debounce protection internally on this input.

**Display Select** — With this input tied to  $V_{CC}$  or at a logic "1", the MM57436 will display the upper 4 digits (16 bits) of the 8 digit (32 bit) counter. Connecting this pin to ground or to a logic "0" will cause the lower 4 digits of the 8 digit counter to be displayed. If the MM57436 is operating as a 4-digit counter (pin 19 open or at  $V_{CC}$ ) the Display Select input is ignored and has no effect whatsoever on the display. This input may be hard wired to either  $V_{CC}$  or ground; may be controlled by a switch or may be controlled by a logic signal. The input may be changed at any time by the user without impairing the operation of the device.

# General Operation

## Initialization

The RESET logic will clear the MM57436 if the power supply rise time is between 1 ms and 1  $\mu$ s. If the power supply rise time is greater than 1 ms, the user must provide an external RC network and diode to the RESET pin as shown below (Figure 2). The RESET input is configured as a Schmitt trigger input. The user may control this with an external signal if desired as long as the proper levels are maintained. The RESET pin is the means by which the user may clear the counter. RESET may be brought low at any time. The MM57436 will be cleared whenever the proper "0" level is applied at the RESET input provided the input stays low for at least 16 clock cycles. If the reset pin is not used it should be connected to  $V_{CC}$ .

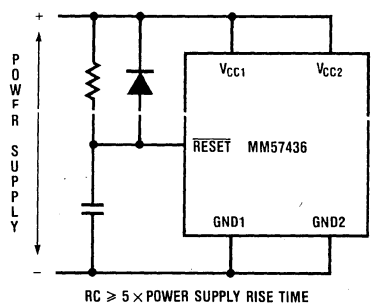


Figure 2. Power-Up Clear Circuit

## Oscillator

The user has the option of connecting an RC network to the OSC IN pin and using the internal oscillator or he may supply an external oscillator to the OSC IN pin. The OSC IN input is a Schmitt trigger input and the user must insure that the proper levels are met when supplying an external clock.

The external oscillator is recommended when the counting speed and/or the stability of the counting speed is critical. The internal RC oscillator is only accurate to about  $\pm 15\%$  to  $\pm 20\%$ . However, if practical in the application, the RC network can be tuned for the desired operating frequency. Some typical RC values that place the operating speed at near the maximum are shown below (Figure 3).

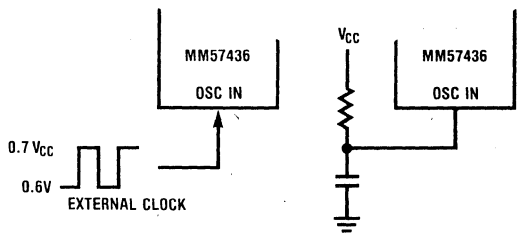
## Power Supply

The MM57436 has two  $V_{CC}$  pins:  $V_{CC1}$  and  $V_{CC2}$  — and two ground pins: GND1 and GND2. Both  $V_{CC1}$  and  $V_{CC2}$  must be connected to the positive supply ( $V_{CC}$ ). Both GND1 and GND2 must be connected to ground. Failure to do this will result in improper operation of the MM57436.

## Count Input

The MM57436 counts negative-going pulses at the Count Input. The width of the negative-going (logic "1" to logic "0") must be at least 8 times the oscillator cycle time.

In order to maximize the counting speed and not to miss any pulses, during the display cycles, the MM57436 has a 4-bit register at the COUNT input which will accumulate up to 15 counts. This register is added/subtracted from the counter. Therefore at the higher input count speeds, when the counter is changed from an up counter to a down counter or vice versa, there is a window of up to 15 counts — the maximum value in the input register — in the count. This effect is completely unobservable at slow input count speeds and gradually becomes more noticeable as the repetition rate of the count pulse increases. If the up/down mode is not changed during operation, the only observable effect of the input register is that the display may appear to increment or decrement by values greater than 1.



RC Controlled Oscillator		
R(k $\Omega$ )	C(pF)	OSC IN Period ( $\mu$ s)
51	100	4.75 $\pm$ 15%
82	56	4.75 $\pm$ 13%

Figure 3. MM57436 Oscillator

## Input/Output Characteristics

### Inputs

The MM57436 has three types of inputs. Figure 4a is the input with a depletion load to  $V_{CC}$  found on pins 17, 18, and 19 (Decimal/Binary, Up/Down, 4/8 Digit). Figure 4b is a slightly different type of input with a depletion load to  $V_{CC}$  found on pins 4 and 14 (RESET, COUNT). The remaining input, pin 5-Display Select, has no load device (Figure 4c).

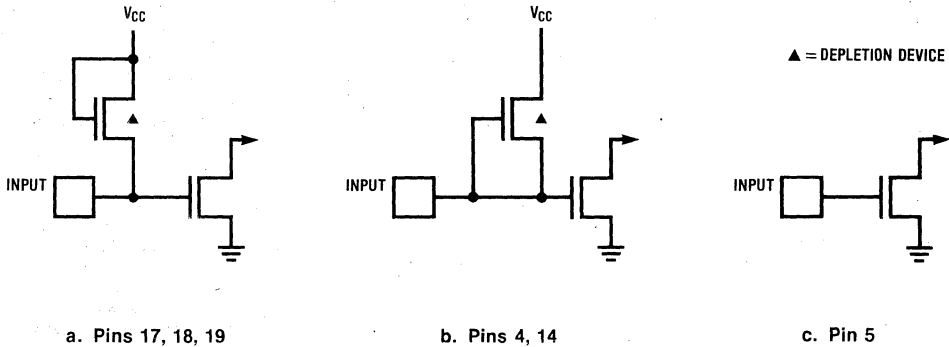


Figure 4. Input Configurations

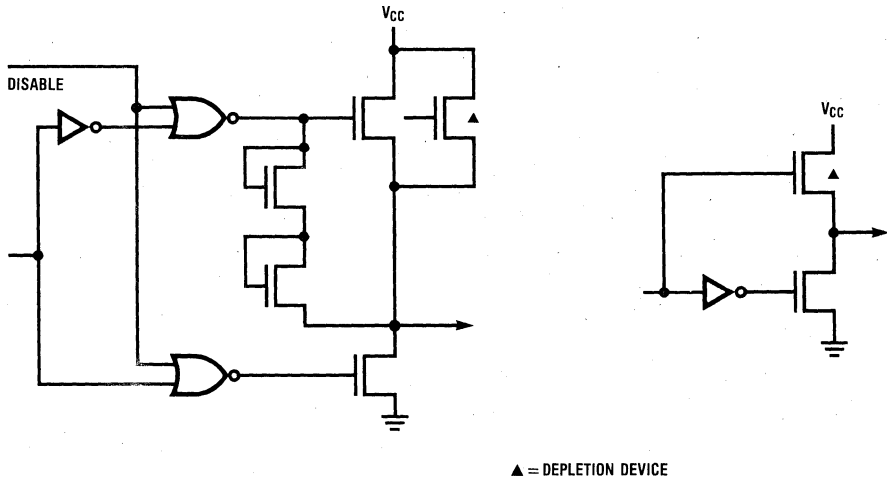


Figure 5. Output Configurations

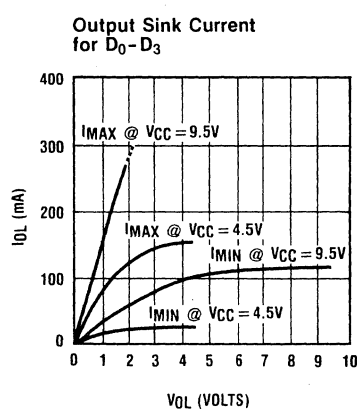
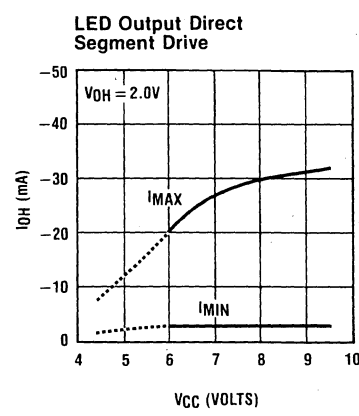
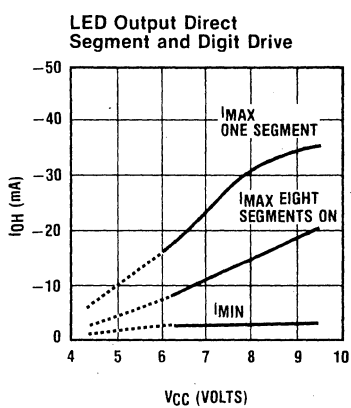
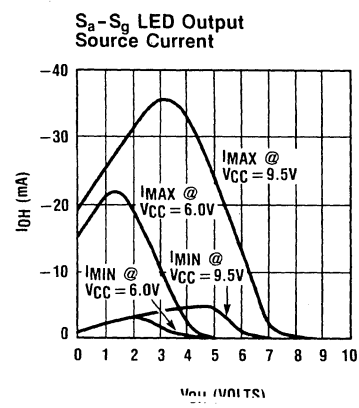
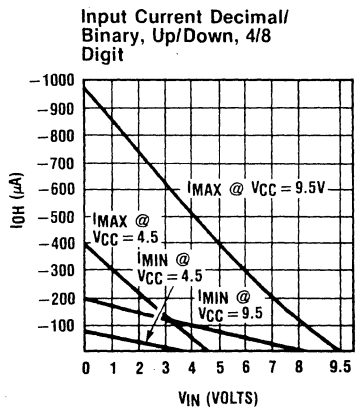
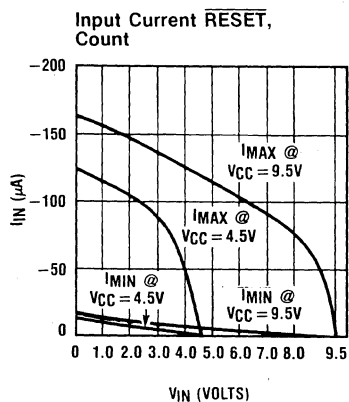


Figure 6. I/O DC Current Characteristics

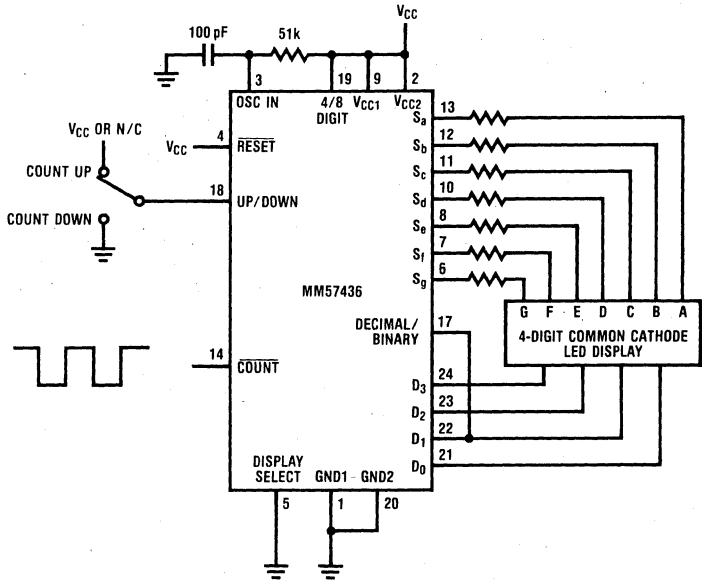


Figure 7. MM57436 as 16-Bit Binary Counter with RC Oscillator and Switch-Controlled Up/Down Mode

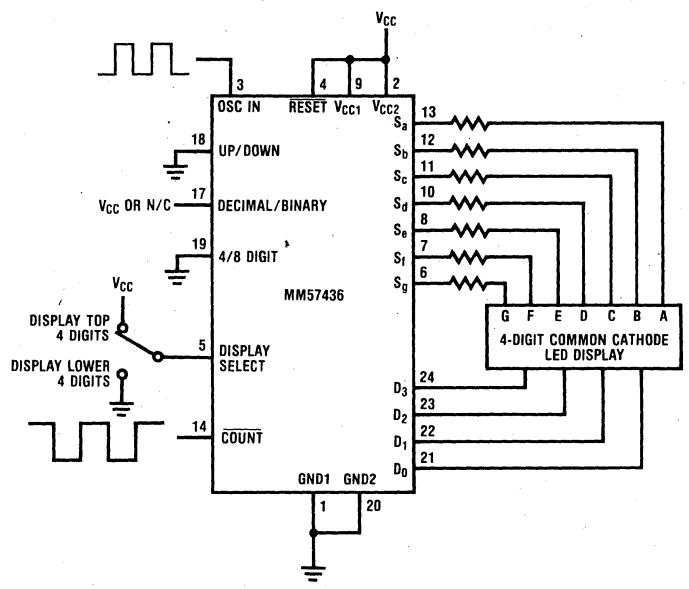


Figure 8. MM57436 as 8-Digit Decimal Down Counter with External Oscillator



# MM57455 Advanced Educational Arithmetic Game

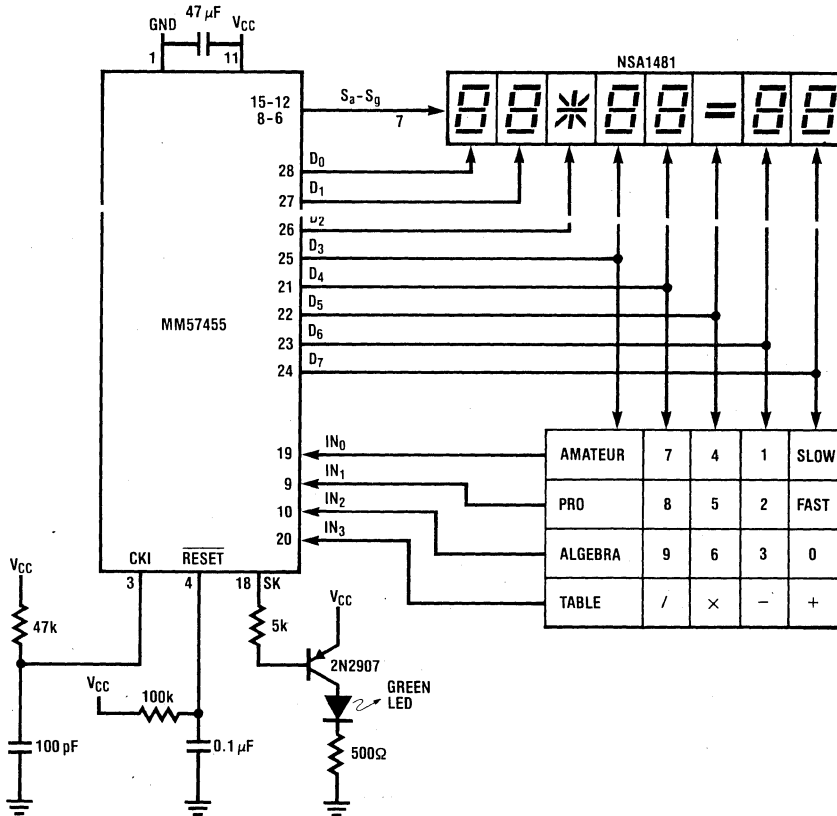
## General Description

Figure 1 contains an electrical diagram of a complete teaching game system.

## Features

- Produces add, subtract, multiply, and divide problems which teach basic arithmetic
- 6,562 different problems are produced
- Problems are generated randomly and automatically
- Automatic entry, no "ENTER" key is needed
- Green LED lights when the correct answer is entered
- If the wrong answer is entered, "E" appears in the display and the user gets a second try
- If the user answers incorrectly on both tries, the correct answer is flashed in the display
- Internal timer gives the user about 10 seconds to answer. If he doesn't answer, the problem is counted wrong
- Ten problems in each problem set
- Number of problems correct appears in the display at the end of a problem set, with the green LED flashing
- "TABLE" button causes non-random problems to be generated
- "COMPLEX" button causes algebra-type problems to be generated
- "AMATEUR/PRO" buttons select easy/hard addition and subtraction problems
- "NORMAL/FAST" buttons select 10 or 3 seconds to answer a problem
- Automatically begins game on power "ON"
- Low system cost (Figure 1)

## Electrical Diagram





## Absolute Maximum Ratings

Voltage at Any Pin Relative to GND1	-0.3V to +10V
Ambient Operating Temperature	0°C to +70°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 Seconds)	300°C
Power Dissipation	0.75 Watt at 25°C 0.4 Watt at 70°C

"Absolute Maximum Ratings" indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.

## DC Electrical Characteristics $0^{\circ}\text{C} \leq T_A \leq 70^{\circ}\text{C}$ , $4.5\text{V} \leq V_{\text{CC}} \leq 9.5\text{V}$ , unless otherwise specified

Parameter	Conditions	Min.	Typ.	Max.	Units
Operating Voltage ( $V_{\text{CC}}$ )		4.5		9.5	V
Operating Supply Current	(all inputs and outputs open)			8	mA
Input Voltage Levels					
OSC IN, RESET					
Logic High ( $V_{\text{IH}}$ )		0.7 $V_{\text{CC}}$			V
Logic Low ( $V_{\text{IL}}$ )				0.6	V
All Other Inputs					
Logic High ( $V_{\text{IH}}$ )	$V_{\text{CC}} = 9.5\text{V}$	3.0			V
Logic High ( $V_{\text{IH}}$ )	$V_{\text{CC}} = 5\text{V} \pm 10\%$	2.0			V
Logic Low ( $V_{\text{IL}}$ )				0.8	V
Output Current Levels					
Output Sink Current					
$D_0$ - $D_7$ ( $I_{\text{OL}}$ )	$V_{\text{CC}} = 9.5\text{V}$ , $V_{\text{OL}} = 1.0\text{V}$	30			mA
	$V_{\text{CC}} = 4.5\text{V}$ , $V_{\text{OL}} = 1.0\text{V}$	15			mA
$S_a$ - $S_g$ ( $I_{\text{OL}}$ )	$V_{\text{CC}} = 9.5\text{V}$ , $V_{\text{OL}} = 0.4\text{V}$	0.8			mA
	$V_{\text{CC}} = 4.5\text{V}$ , $V_{\text{OL}} = 0.4\text{V}$	0.4			mA
Output Source Current					
$S_a$ - $S_g$ ( $I_{\text{OH}}$ )	$V_{\text{CC}} = 9.5\text{V}$ , $V_{\text{OH}} = 2.0\text{V}$	-3.0		-35	mA
	$V_{\text{CC}} = 6.0\text{V}$ , $V_{\text{OH}} = 2.0\text{V}$	-3.0		-25	mA

## Functional Description

### Display Configuration

The special LED display used with the MM57455 displays any of the 4 symbols "+", "-", "x", "p" in the third digit position. An "=" is displayed in the sixth digit position. The remaining 6 digits are normal 7-segment numeral displays.

### Power "ON"

Upon powering "ON" the MM57455, it begins displaying the symbols "+", "-", "x", "p", "+", ... one after another, each lasting about 1/2 second. This indicates that it is at the beginning of a "problem set" and ready to accept a function key input.

### Key Operations

#### Function Keys, "+", "-", "x", "p"

One of these keys is depressed to begin a problem set. After pressing one of these keys, a randomly generated problem appears in the display. The problem is either "+", "-", "x", "p", depending on the key that was pressed.

### Number Keys, "0-9"

These keys are used to enter answers to problems. After a problem appears in the display, the user has 2 tries to answer it correctly.

### Green LED

If the user keys in the correct answer to a problem, the green LED lights up immediately for 1 1/2 seconds. Then a new problem appears.

### Incorrect Answer Indicator

If the user keys in a wrong answer to a problem, his answer disappears in the display and an "E" appears.

### Second Try

If the user answers incorrectly, he gets a second try. When the "E" appears (indicating that the answer is wrong), he types in his second try. Again, the green LED lights if correct, and an "E" appears if wrong.

## Functional Description (cont'd)

### Internal Timer

The MM57455 has an internal timer which allows the user 10 seconds to answer a problem. If he doesn't answer in 10 seconds, an "E" appears in the display, indicating a wrong answer. The user then gets a second try and again must answer within 10 seconds.

### Flashing of a Correct Answer

In the user answers wrong on both tries, the correct answer flashes in the display. Then the next problem appears.

### Ten Problems per Problem Set

New problems appear one after another until 10 problems have been done.

### Score at End of Problem Set

After 10 problems are done, the number of problems the user got right appears in the display, and the green LED flashes. Only first try answers are counted correct. After 16 flashes, the MM57455 again displays "+", "-", "x", "/", "+", ... and is ready for another function key entry.

### "TABLE" Key

If the "TABLE" key is depressed just before pressing a function key at the start of a problem set, table problems will appear, with a random table digit.

**Example:** press "TABLE" x  
and these problems may appear:

$$6 \times 1 =$$

$$6 \times 2 =$$

$$6 \times 3 =$$

.

.

$$6 \times 10 =$$

A non-random table digit can be selected by depressing the desired number (1-10) just before pressing a function button at the start of a problem set.

**Example:** press 9 x  
and these problems will appear:

$$9 \times 1 =$$

$$9 \times 2 =$$

$$9 \times 3 =$$

.

.

$$9 \times 0 =$$

### "ALGEBRA" Key

If the "ALGEBRA" key is depressed just before pressing a function key at the start of a problem set, algebra-type problems will be displayed (the answer is present and one of the factors is blank, as:  $(15 + \quad = 21)$ ). The user must enter the missing factor. (Note. Both "ALGEBRA" and "TABLE" buttons may be pressed before pressing a function key. This will cause algebra-type table problems to be displayed.) The order of depression is unimportant; i.e., "ALGEBRA" or "TABLE" may be pressed first.

### "AMATEUR/PRO" Keys

These keys select easy ("AMATEUR") or hard ("PRO") addition and subtraction problems. Easy means sum  $< 30$  and difference  $< 20$ . Hard means sum  $< 100$  and difference  $< 100$ .

When power is turned "ON", the machine is in easy ("AMATEUR") mode.

### "NORMAL/FAST" Keys

These keys are used to select 10 second ("NORMAL") or 3 second ("FAST") answer time.

When power is turned "ON", the machine is in the 10 second ("NORMAL") mode.

## MM57459 8-Digit LED Direct-Drive Memory Calculator

### General Description

The single-chip MM57459 calculator was developed using an N-channel enhancement and depletion mode MOS/LSI technology with a primary object of low end-product cost. A complete calculator as shown in *Figure 1* requires only the MM57459 calculator chip, and X-Y matrix keyboard, an NSA1188 LED display and a 9V battery.

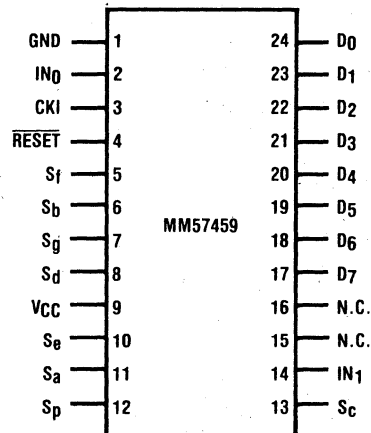
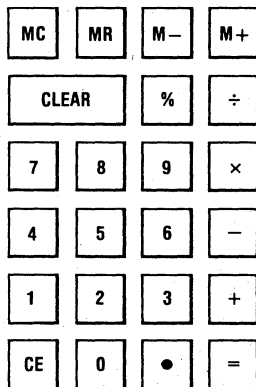
Keyboard decoding and key debounce circuitry, all clocks, and timing generators, power-on clear, and 7-segment output display decoding are included on-chip, and require no external components. Segments and digits can usually be driven directly from the MM57459, as the segments source up to 30mA max. peak current and the digit drivers sink 30mA min.

Leading zero suppression and a floating negative sign allow convenient reading of the display and conserve power. Up to 8 digits for positive numbers and 7 for negative numbers can be displayed, with the negative sign displayed in the left-most position.

### Features

- 8 Digits with four key memory (M+, M-, MR, MC)
- Low voltage operation (single power supply)
- Direct interface with digits and segments of LED display
- Percent function with add-on/discount
- Automatic constant on all five functions
- Floating minus sign
- Leading zero suppression
- Internal clock generator
- Internal encoding for keyboard inputs
- Internal debouncing for keyboard inputs
- Display flash in calculator overflow state

### Typical Keyboard and Connection Diagram



Top View

Order Number MM57459N  
NS Package N24A

## Absolute Maximum Ratings

Voltage at Any Pin Relative to GND1	-0.3V to +10V
Ambient Operating Temperature	0°C to +70°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 Seconds)	300°C
Power Dissipation	0.75 Watt at 25°C 0.4 Watt at 70°C

*Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

## DC Electrical Characteristics $0^{\circ}\text{C} \leq T_A \leq 70^{\circ}\text{C}$ , $4.5\text{V} \leq V_{CC} \leq 9.5\text{V}$ , unless otherwise specified

Parameter	Conditions	Min.	Typ.	Max.	Units
Operating Voltage ( $V_{CC}$ )		4.5		9.5	V
Operating Supply Current	(all inputs and outputs open)			8	mA
input Voltage Levels CKI, RESET					
Logic High ( $V_{IH}$ )		$0.7V_{CC}$			V
Logic Low ( $V_{IL}$ )				0.6	V
All Other Inputs					
Logic High ( $V_{IH}$ )	$V_{CC} = 9.5\text{V}$	3.0			V
Logic High ( $V_{IH}$ )	$V_{CC} = 5\text{V} \pm 10\%$	2.0			V
Logic Low ( $V_{IL}$ )				0.8	V
Output Current Levels					
Output Sink Current $D_0-D_3$ ( $I_{OL}$ )	$V_{CC} = 9.5\text{V}$ , $V_{OL} = 1.0\text{V}$	30			mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 1.0\text{V}$	15			mA
$S_a-S_g$ , $S_p$ ( $I_{OL}$ )	$V_{CC} = 9.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.8			mA
	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.4			mA
Output Source Current $S_a-S_g$ , $S_p$ ( $I_{OH}$ )	$V_{CC} = 9.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-3.0		-35	mA
	$V_{CC} = 6.0\text{V}$ , $V_{OH} = 2.0\text{V}$	-3.0		-25	mA

## 1. Key Definition

0 - 9 -  $\square$

The first number key in a sequence will clear the display and enter the digit in the LSD of the display. Successive entries will shift the display left and enter data in the LSD. The first decimal point entered is effective. An attempted entry of more than 8 digits or 7 decimal places will be ignored.

$\square$  C — Clear

Clears the display and constant registers, and the result overflow indicator. Memory register is not affected by key. In the memory overflow condition, this key is operative as a clear memory key.

$\square$  CE — Clear Entry

Clears the display of a number entry. In the result overflow mode, this key resets the overflow condition and allows calculation to continue; however this key is inoperative during memory overflow.

$\square$  MC — Memory Clear

Clears the memory.

$\square$  + — Plus

Stores an addition operation and performs a possible preceding operation. Successive depression of the plus key will not affect the display.

$\square$  - — Minus

Stores a subtract operation and performs a possible preceding operation. Repeat subtraction by the minus key will not be possible. If this is depressed after a % , + , or = key, subtraction becomes the pending operation. Immediately following a  $\times$  or  $\div$  key, this acts as a data entry and -0. is displayed.

$\square$   $\times$  — Multiply

Operates the same as the plus key except that a multiply command is stored. Successive depression of the multiply key will not alter the display.

$\square$   $\div$  — Divide

Operates the same as the plus key except that a divide command is stored. Successive depression of the divide key will not alter the display.

$\square$  = — Equal

Executes any previous operation and maintains that operation for possible use in the implied constant mode. The first factor entered for multiplication and the second factor entered for division, subtraction, and addition, are retained for the constant operation. Completes the add-on or discount mode when used following the % key. The first depression of the equal key immediately following a + or - key will not alter the display.

$\square$  % — Percent

The purpose of the percent key is to allow for the calculation of add-on and discount. Determination of add-on requires the principal amount to be the first enter followed by the + or  $\times$  key, with the percentage being the second entry. Depression of the percent key yields the amount to be added-on, such as tax or interest. Depression of the = key adds this amount to be principal. Discount is determined in a similar manner using the - key ( $\times$  and - keys). In the constant mode, new percentages to be added-on may be entered while retaining the principal amount.

$\square$  MR — Memory Recall

Transfers the contents of the memory register into the display register. Memory is retained except in the memory overflow condition. In this case, memory is cleared and its previous contents are displayed in the result overflow mode.

$\square$  M+ — Memory Plus

Add the current display to the contents of memory. M+ will terminate a number entry.

$\square$  M- — Memory Minus

Subtracts current display from the contents of memory. M- will terminate a number entry.

## 2. Error Conditions

### Result Overflow

If the result in absolute value exceeds  $10^8 - 1$ , the display will flash, and only the C and CE keys are operative.

### Memory Overflow

If a M+ or M- operation causes the contents of memory to exceed the above value, the display will flash. In this overflow condition, only the C key is operative.

## 3. Operation Characteristics

### Data Entry

Entry is always floating. On data entry, the data will be right hand justified with the last digit entered always appearing in the least significant digit position. The display register will left shift the display one digit as each new digit is entered.

### Data Output

The output data as a result of a calculation will be right hand justified such that trailing insignificant zeros after the decimal are not displayed. Numbers less than one (1) will be displayed with one leading zero (0.25 for example). Numbers greater than one (1) will not display zeros to the left of the most significant digit.

**Output Display**

The output segments are fully decoded for standard seven-segment display. The digit outputs are multiplexed with the segment scan to provide the output.

**Digit and Segment Buffers**

The segment buffers provide constant drop and operate in conjunction with the constant current digit buffers to provide display current.

**Constant Operation**

The MM57459 has an implied constant mode of operation on +, -, x, ÷, and % operations. The constant calculation is performed automatically by the = key, % key, or % = keys without a constant switch. The second operand is treated as the constant for add, subtract, and divide and the first operand is the constant for multiplication.

For A ± B%-type calculations, the first operand is treated as the constant with the percentage displayed with the proper sign.

**Decimal Alignment**

The results of addition or subtraction remain aligned to the preceding entry having the most decimal places unless a right shift is needed to keep the eight most significant digits (in which case the least significant decimal digits are lost).

**Display Font**

The following table shows the required segment outputs as a function of the display. In the truth table, the symbol • is used to indicate a selected segment.

Character	Display	SA	SB	SC	SD	SE	SF	SG	SP
0	0	•	•	•	•	•	•		
1	1		•	•					
2	2	•	•		•	•		•	
3	3	•	•	•	•			•	
4	4		•	•			•	•	
5	5	•		•	•		•	•	
6	6	•		•	•	•	•	•	
7	7	•	•						
8	8	•	•	•	•	•	•	•	
9	9	•	•	•	•	•	•	•	
Minus Sign	-								•
Dec. Pt.	.								•

RESULT OVf: THE DISPLAY WILL FLASH.  
MEMORY OVf: THE DISPLAY WILL FLASH.

**Floating Minus Sign**

When displaying a negative number the minus indication will be located one digit to the left of the MSD display.

The results of multiplication and division are completely right justified such that only the most significant digits are displayed (the digits not displayed will be truncated).The C key resets decimal alignment.

**Successive Operations**

Only the last operation entered is performed unless a - entry follows a x or ÷ which sets up the calculator for numeric entry only.

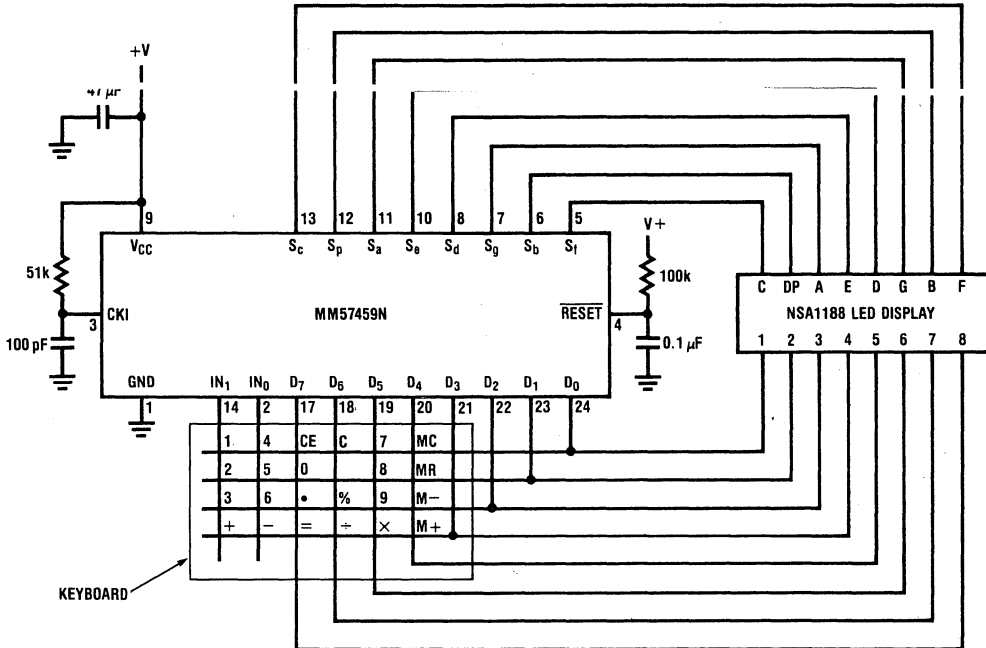
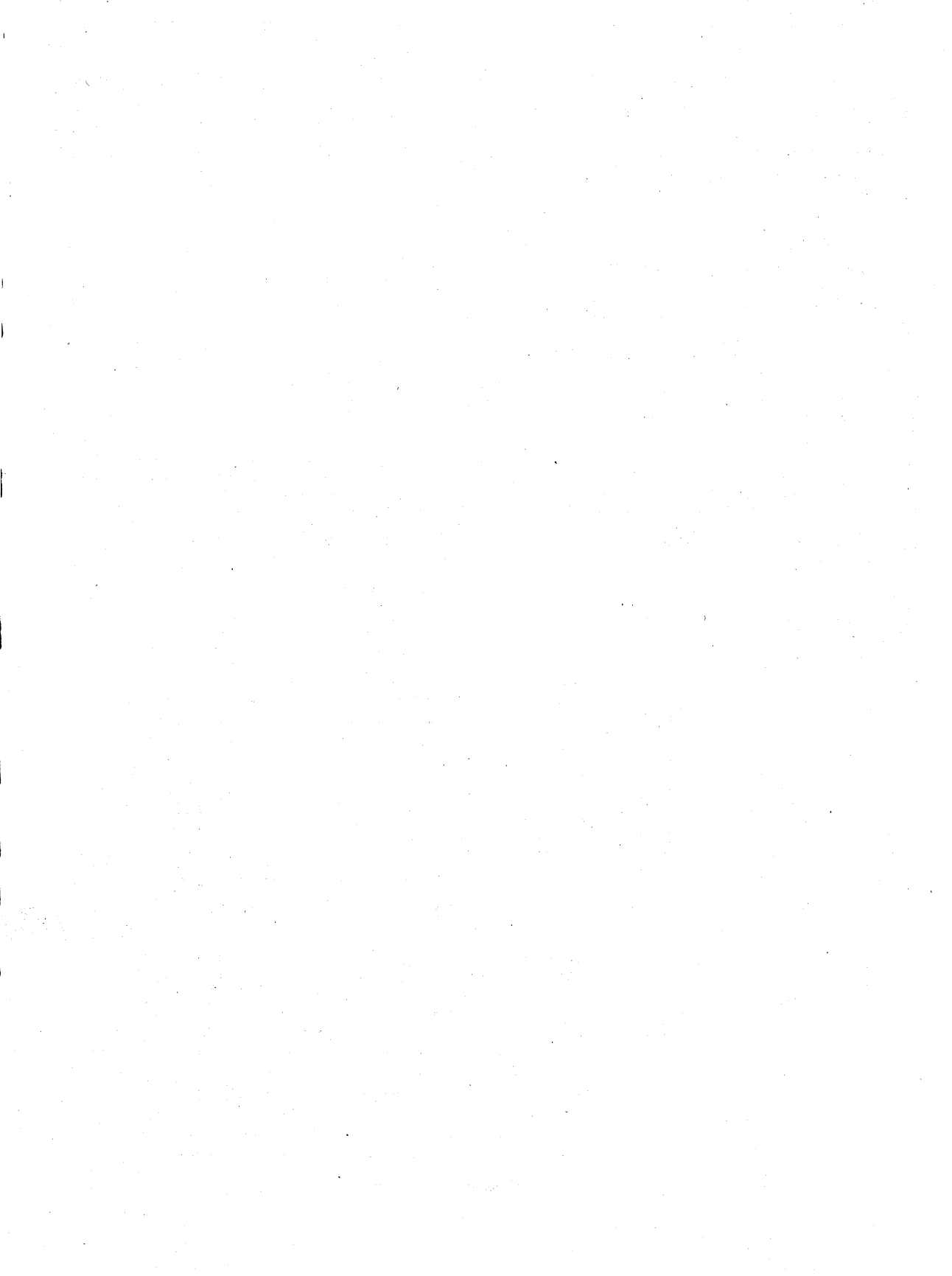


Figure 1. Typical Calculator Application

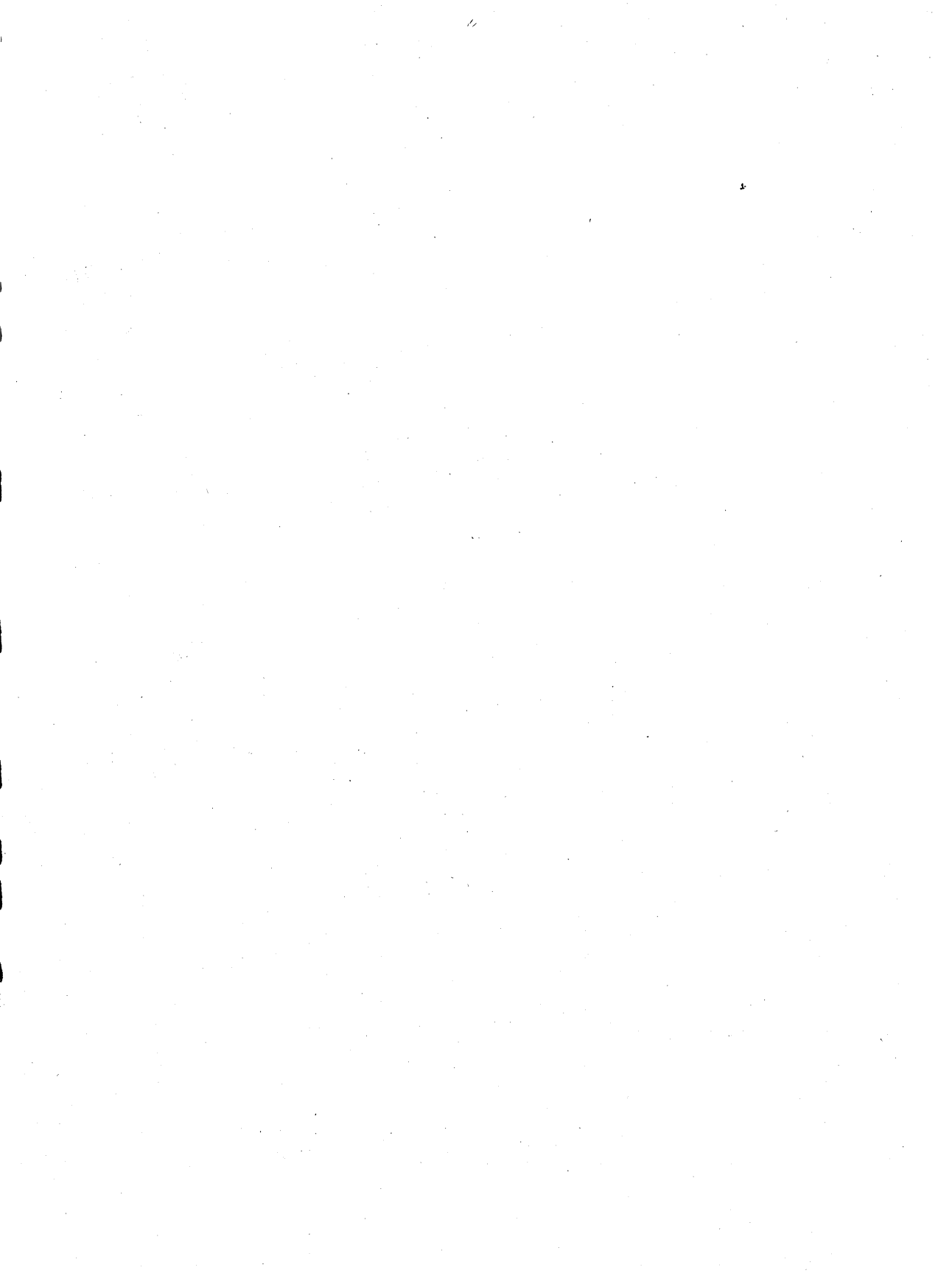




**Section 7**  
**EPROMs and**  
**Support Circuits**







# MM2716 16,384-Bit (2048 × 8) UV Erasable PROM

## General Description

The MM2716 is a high speed 16k UV erasable and electrically reprogrammable EPROM ideally suited for applications where fast turn-around and pattern experimentation are important requirements.

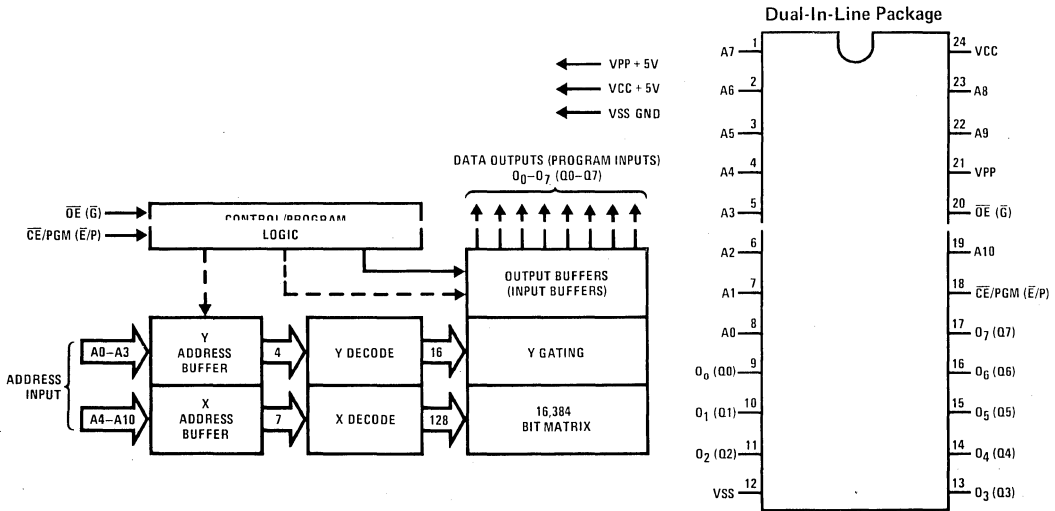
The MM2716 is packaged in a 24-pin dual-in-line package with transparent lid. The transparent lid allows the user to expose the chip to ultraviolet light to erase the bit pattern. A new pattern can then be written into the device by following the programming procedure.

This EPROM is fabricated with the reliable, high volume, time proven, N-channel silicon gate technology.

## Features

- 2048 × 8 organization
- 525 mW max active power, 132 mW max standby power
- Low power during programming
- Access time—MM2716, 450 ns; MM2716-1, 350 ns; MM2716-2, 390 ns
- Single 5V power supply
- Static—no clocks required
- Inputs and outputs TTL compatible during both read and program modes
- TRI-STATE® output

## Block and Connection Diagrams \*



TOP VIEW  
Order Number MM2716Q, MM2716Q-1  
or MM2716Q-2  
See NS Package J24CQ

Pin Connection During Read or Program

MODE	PIN NAME/NUMBER				
	$\overline{\text{CE}}/\text{PGM}$ (E/P) 18	$\overline{\text{OE}}$ (G) 20	VPP 21	VCC 24	OUTPUTS 9-11, 13-17
Read	VIL	VIL	5	5	DOUT
Program	Pulsed VIL to VIH	VIH	25	5	DIN

### Pin Names

A0-A10	Address Inputs
O0-O7 (Q0-Q7)	Data Outputs
$\overline{\text{CE}}/\text{PGM}$ (E/P)	Chip Enable/Program
$\overline{\text{OE}}$ (G)	Output Enable
VPP	Read 5V, Program 25V
VCC	Power (5V)
VSS	Ground

\*Symbols in parentheses are proposed industry standard

## Absolute Maximum Ratings (Note 1)

Temperature Under Bias	-25°C to +85°C
Storage Temperature	-65°C to +125°C
VPP Supply Voltage with Respect to VSS	26.5V to -0.3V

All Input or Output Voltages with Respect to VSS (except VPP)	6V to -0.3V
Power Dissipation	1.5 W
Lead Temperature (Soldering, 10 seconds)	300°C

## READ OPERATION (Note 2)

### DC Operating Characteristics

$T_A = 0^\circ\text{C}$  to  $+70^\circ\text{C}$ ,  $V_{CC} = 5V \pm 5\%$ , ( $V_{CC} = 5V \pm 10\%$  for MM2716-1),  
 $V_{PP} = V_{CC} \pm 0.6V$  (Note 3),  $V_{SS} = 0V$ , unless otherwise noted.

SYMBOL	PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
ILI	Input Current	$V_{IN} = 5.25V$ or $V_{IN} = V_{IL}$			10	$\mu\text{A}$
ILO	Output Leakage Current	$V_{OUT} = 5.25V$ , $\overline{CE}/PGM = 5V$			10	$\mu\text{A}$
IPP1	VPP Supply Current	$V_{PP} = 5.85V$			5	mA
ICC1	VCC Supply Current (Standby)	$\overline{CE}/PGM = V_{IH}$ , $\overline{OE} = V_{IL}$		10	25	mA
ICC2	VCC Supply Current (Active)	$\overline{CE}/PGM = \overline{OE} = V_{IL}$		57	100	mA
VIL	Input Low Voltage		0.1		0.8	V
VIH	Input High Voltage		2.0		$V_{CC} + 1$	V
VOH	Output High Voltage	$I_{OH} = 400 \mu\text{A}$	2.4			V
VOL	Output Low Voltage	$I_{OL} = 2.1 \text{ mA}$			0.45	V

## AC Characteristics (Note 4)

$T_A = 0^\circ\text{C}$  to  $+70^\circ\text{C}$ ,  $V_{CC} = 5V \pm 5\%$ , ( $V_{CC} = 5V \pm 10\%$  for MM2716-1),  
 $V_{PP} = V_{CC} \pm 0.6V$  (Note 3),  $V_{SS} = 0V$ , unless otherwise noted.

SYMBOL		PARAMETER	CONDITIONS	MM2716		MM2716-1		MM2716-2		UNITS
ALTERNATE	STANDARD			MIN	MAX	MIN	MAX	MIN	MAX	
$t_{ACC}$	TAVQV	Address to Output Delay	$\overline{CE}/PGM = \overline{OE} = V_{IL}$		450		350		390	ns
$t_{CE}$	TELQV	$\overline{CE}$ to Output Delay	$\overline{OE} = V_{IL}$		450		350		390	ns
$t_{OE}$	TGLQV	Output Enable to Output Delay	$\overline{CE}/PGM = V_{IL}$		120		120		120	ns
$t_{DF}$	TGHQZ	Output Enable High to Output Hi-Z	$\overline{CE}/PGM = V_{IL}$	0	100	0	100	0	100	ns
$t_{OH}$	TAXQX	Address to Output Hold	$\overline{CE}/PGM = \overline{OE} = V_{IL}$	0		0		0		ns
$t_{OD}$	TEHQZ	$\overline{CE}$ to Output Hi-Z	$\overline{OE} = V_{IL}$	0	100	0	100	0	100	ns

## Capacitance (Note 5)

$T_A = 25^\circ\text{C}$ ,  $f = 1 \text{ MHz}$

SYMBOL	PARAMETER	CONDITIONS	TYP	MAX	UNITS
CI	Input Capacitance	$V_{IN} = 0V$	4	6	pF
CO	Output Capacitance	$V_{OUT} = 0V$	8	12	pF

**Note 1:** "Absolute Maximum Ratings" are those values beyond which the safety of the device cannot be guaranteed. Except for "Operating Temperature Range" they are not meant to imply that the devices should be operated at these limits. The table of "Electrical Characteristics" provides conditions for actual device operation.

**Note 2:** Typical conditions are for operation at:  $T_A = 25^\circ\text{C}$ ,  $V_{CC} = 5V$ ,  $V_{PP} = V_{CC}$ , and  $V_{SS} = 0V$ .

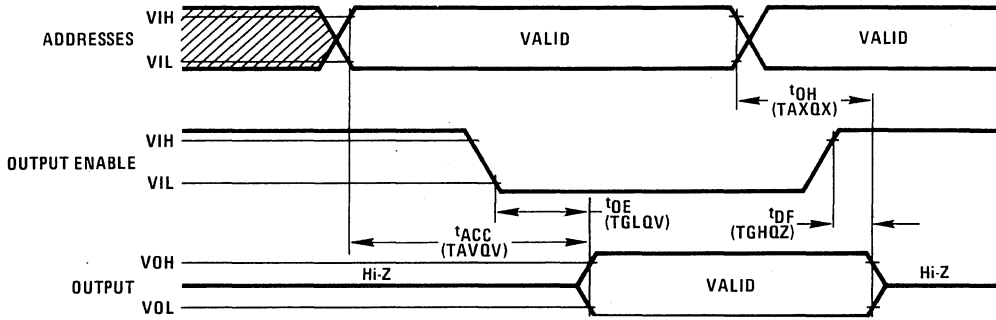
**Note 3:** VPP may be connected to VCC except during program. The  $\pm 0.6V$  tolerance allows a circuit to switch VPP between the read voltage and the program voltage.

**Note 4:** Output load: 1 TTL gate and  $CL = 100 \text{ pF}$ . Input rise and fall times  $\leq 20 \text{ ns}$ .

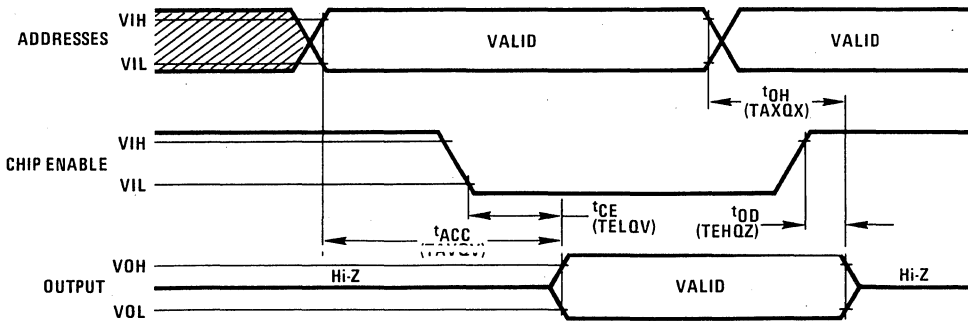
**Note 5:** Capacitance is guaranteed by periodic testing.

Switching Time Waveforms \*

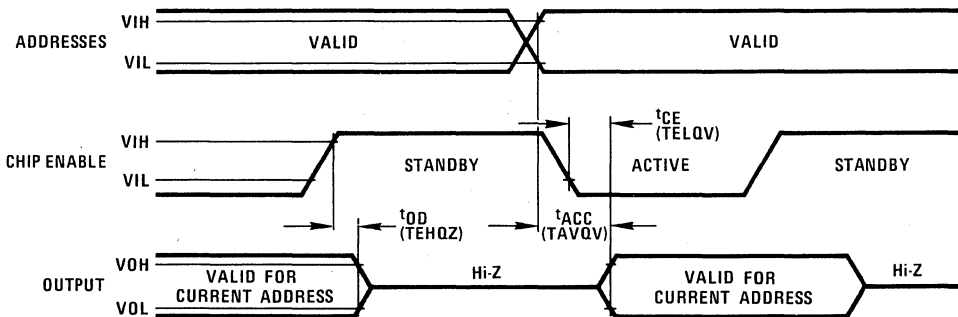
Read Cycle ( $\overline{CE}/PGM = VIL$ )



Read Cycle ( $\overline{OE} = VIL$ )



Standby Power Down Mode ( $\overline{OE} = VIL$ )



\*Symbols in parentheses are proposed industry standard

## PROGRAM OPERATION

### DC Electrical Characteristics and Operating Conditions (Notes 1 and 2)

( $T_A = 25^\circ\text{C} \pm 5^\circ\text{C}$ ) ( $V_{CC} = 5\text{V} \pm 5\%$ ,  $V_{PP} = 25\text{V} \pm 1\text{V}$ )

SYMBOL	PARAMETER	MIN	TYP	MAX	UNITS
ILI	Input Leakage Current (Note 3)			10	$\mu\text{A}$
VIL	Input Low Level	-0.1		0.8	V
VIH	Input High Level	2.0		$V_{CC} + 1$	V
ICC	VCC Power Supply Current			100	mA
IPP1	VPP Supply Current (Note 4)			5	mA
IPP2	VPP Supply Current During Programming Pulse (Note 5)			30	mA

### AC Characteristics and Operating Conditions (Notes 1, 2, and 6)

( $T_A = 25^\circ\text{C} \pm 5^\circ\text{C}$ ) ( $V_{CC} = 5\text{V} \pm 5\%$ ,  $V_{PP} = 25\text{V} \pm 1\text{V}$ )

SYMBOL		PARAMETER	MIN	TYP	MAX	UNITS
ALTERNATE	STANDARD					
tAS	TAVPH	Address Setup Time	2			$\mu\text{s}$
tOS	TGHPH	$\overline{\text{OE}}$ Setup Time	2			$\mu\text{s}$
tDS	TDVPH	Data Setup Time	2			$\mu\text{s}$
tAH	TPLAX	Address Hold Time	2			$\mu\text{s}$
tOH	TPLGX	$\overline{\text{OE}}$ Hold Time	2			$\mu\text{s}$
tDH	TPLDX	Data Hold Time	2			$\mu\text{s}$
tDF	TGHQZ	Chip Disable to Output Float Delay (Note 4)	0		100	ns
tCE	TGLQV	Chip Enable to Output Delay (Note 4)			120	ns
tpW	TPHPL	Program Pulse Width	45	50	55	ms
tpR	TPH1PH2	Program Pulse Rise Time	5			ns
tpF	TPL2PL1	Program Pulse Fall Time	5			ns

**Note 1:** VCC must be applied at the same time or before VPP and removed after or at the same time as VPP. To prevent damage to the device it must not be inserted into a board with power applied.

**Note 2:** Care must be taken to prevent overshoot of the VPP supply when switching to +25V.

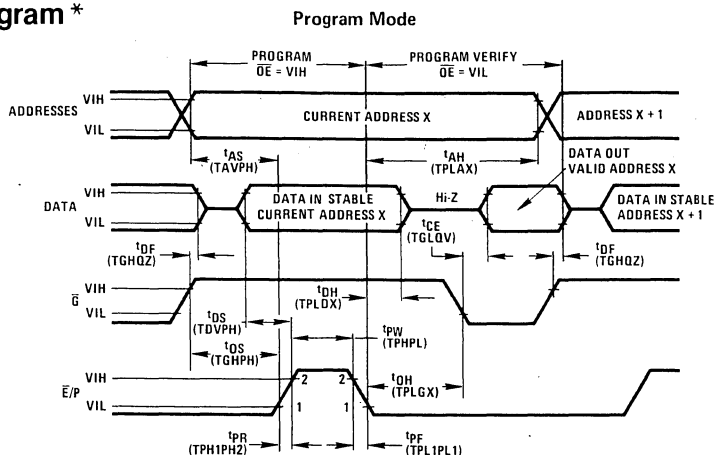
**Note 3:**  $0.45\text{V} \leq V_{IN} \leq 5.25\text{V}$ .

**Note 4:**  $\overline{\text{CE}}/\text{PGM} = V_{IL}$ ,  $V_{PP} = V_{CC} + 0.6\text{V}$ .

**Note 5:**  $V_{PP} = 26\text{V}$ .

**Note 6:** Transition times  $\leq 20$  ns unless noted otherwise.

## Timing Diagram \*



## Functional Description

## DEVICE OPERATION

The MM2716 has 3 modes of operation in the normal system environment. These are shown in Table I.

## Read Mode

The MM2716 read operation requires that  $\overline{OE} = VIL$ ,  $\overline{CE}/PGM = VIL$  and that addresses A0–A10 have been stabilized. Valid data will appear on the output pins after  $t_{ACC}$ ,  $t_{OE}$  or  $t_{CE}$  times (see Switching Time Waveforms) depending on which is limiting.

## Deselect Mode

The MM2716 is deselected by making  $\overline{OE} = VIH$ . This mode is independent of  $\overline{CE}/PGM$  and the condition of the addresses. The outputs are Hi-Z when  $\overline{OE} = VIH$ . This allows OR-tying 2 or more MM2716's for memory expansion.

## Standby Mode (Power Down)

The MM2716 may be powered down to the standby mode by making  $\overline{CE}/PGM = VIH$ . This is independent of  $\overline{OE}$  and automatically puts the outputs in their Hi-Z state. The power is reduced to 25% (132 mW max) of the normal operating power. VCC and VPP must be maintained at 5V. Access time at power up remains either  $t_{ACC}$  or  $t_{CE}$  (see Switching Time Waveforms).

## PROGRAMMING

The MM2716 is shipped from National completely erased. All bits will be at a "1" level (output high) in this initial state and after any full erasure. Table II shows the 3 programming modes.

TABLE I. OPERATING MODES (VCC = VPP = 5V)

MODE	PIN NAME/NUMBER		
	$\overline{CE}/PGM$ ( $\overline{E}/P$ ) 18	$\overline{OE}$ ( $\overline{G}$ ) 20	OUTPUTS 9–11, 13–17
Read	VIL	VIL	DOUT
Deselect	Don't Care	VIH	Hi-Z
Standby	VIH	Don't Care	Hi-Z

TABLE II. PROGRAMMING MODES (VCC = 5V)

MODE	PIN NAME/NUMBER			
	$\overline{CE}/PGM$ ( $\overline{E}/P$ ) 18	$\overline{OE}$ ( $\overline{G}$ ) 20	VPP 21	OUTPUTS Q 9–11, 13–17
Program	Pulsed VIL to VIH	VIH	25	DIN
Program Verify	VIL	VIL	25(5)	DOUT
Program Inhibit	VIL	VIH	25	Hi-Z

\*Symbols in parentheses are proposed industry standard

## Functional Description (Continued)

### Program Mode

The MM2716 is programmed by introducing "0's into the desired locations. This is done 8 bits (a byte) at a time. Any individual address, a sequence of addresses, or addresses chosen at random may be programmed. Any or all of the 8 bits associated with an address location may be programmed with a single program pulse applied to the chip enable pin. All input voltage levels, including the program pulse on chip enable are TTL compatible. The programming sequence is:

With  $V_{PP} = 25V$ ,  $V_{CC} = 5V$ ,  $\overline{OE} = V_{IH}$  and  $\overline{CE}/PGM = V_{IL}$ , an address is selected and the desired data word is applied to the output pins. ( $V_{IL} = "0"$  and  $V_{IL} = "1"$  for both address and data.) After the address and data signals are stable the program pin is pulsed from  $V_{IL}$  to  $V_{IH}$  with a pulse width between 45 ms and 55 ms.

Multiple pulses are not needed but will not cause device damage. No pins should be left open. A high level ( $V_{IH}$  or higher) *must not* be maintained longer than  $tpw(MAX)$  on the program pin during programming. MM2716's may be programmed in parallel with the same data in this mode.

### Program Verify Mode

The programming of the MM2716 may be verified either 1 word at a time during the programming (as shown in the timing diagram) or by reading all of the words out at the end of the programming sequence. This can be done with  $V_{PP} = 25V$  (or 5V) in either case.

### Program Inhibit Mode

The program inhibit mode allows programming several MM2716's simultaneously with different data for each one by controlling which ones receive the program pulse. All similar inputs of the MM2716 may be paralleled. Pulsing the program pin (from  $V_{IL}$  to  $V_{IH}$ ) will program

a unit while inhibiting the program pulse to a unit will keep it from being programmed and keeping  $\overline{OE} = V_{IH}$  will put its outputs in the Hi-Z state.

### ERASING

The MM2716 is erased by exposure to high intensity ultraviolet light through the transparent window. This exposure discharges the floating gate to its initial state through induced photo current. It is recommended that the MM2716 be kept out of direct sunlight. The UV content of sunlight may cause a partial erasure of some bits in a relatively short period of time. Direct sunlight can also cause temporary functional failure. Extended exposure to room level fluorescent lighting will also cause erasure. An opaque coating (paint, tape, label, etc.) should be placed over the package window if this product is to be operated under these lighting conditions.

An ultraviolet source of 2537 Å yielding a total integrated dosage of 15 watt-seconds/cm<sup>2</sup> is required. This will erase the part in approximately 15 to 20 minutes if a UV lamp with a 12,000 μW/cm<sup>2</sup> power rating is used. The MM2716 to be erased should be placed 1 inch away from the lamp and no filters should be used.

An erasure system should be calibrated periodically. The distance from lamp to unit should be maintained at 1 inch. The erasure time is increased by the square of the distance (if the distance is doubled the erasure time goes up by a factor of 4). Lamps lose intensity as they age. When a lamp is changed, the distance is changed, or the lamp is aged, the system should be checked to make certain full erasure is occurring. Incomplete erasure will cause symptoms that can be misleading. Programmers, components, and system designs have been erroneously suspected when incomplete erasure was the basic problem.

# NMC27C16 16,384-Bit (2048 × 8) UV Erasable CMOS PROM

Parameter/Part Number	NMC27C16Q-45	NMC27C16Q-55	NMC27C16Q-65
Access Time (ns)	450	550	650
Active Current (mA)	5	5	5
Standby Current (mA)	0.1	0.1	0.1

## General Description

The NMC27C16 is a high speed 16k UV erasable and electrically reprogrammable CMOS EPROM ideally suited for applications where fast turn-around, pattern experimentation and low power consumption are important requirements.

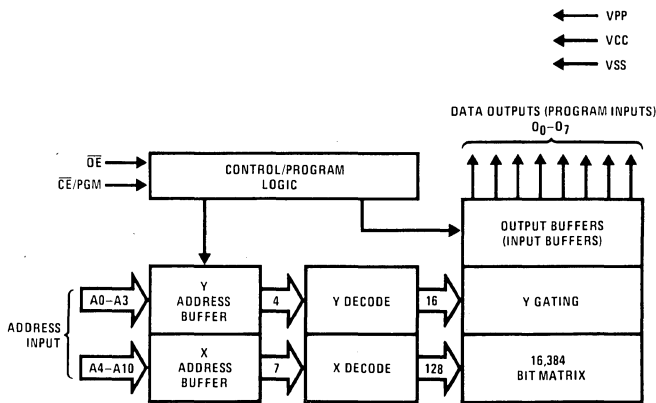
The NMC27C16 is packaged in a 24 pin dual in line package with transparent lid. The transparent lid allows the user to expose the chip to ultraviolet light to erase the bit pattern. A new pattern can then be written into the device by following the programming procedure.

This EPROM is fabricated with the reliable, high volume, time proven, P<sup>2</sup>CMOS silicon gate technology.

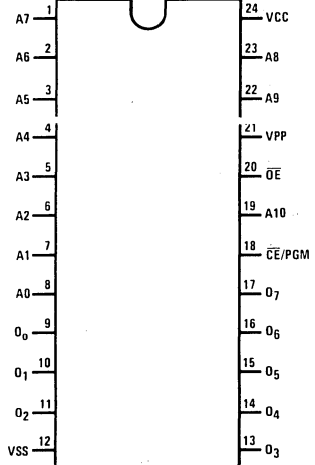
## Features

- CMOS power consumption  
53 mW max active  
5.3 mW max standby
- Performance compatible to NSC800 CMOS microprocessor and NMC6716 synchronous CMOS EPROM
- 2048 × 8 organization
- Pin compatible to 2716
- Access time down to 450 ns
- Single 5V power supply
- Static—no clocks required
- Inputs and outputs TTL compatible during both read and program modes
- TRI-STATE<sup>®</sup> output

## Block and Connection Diagrams



### Dual-In-Line Package



### Pin Connection During Read or Program

Mode	Pin Name/Number				
	CE/PGM 18	OE 20	VPP 21	VCC 24	Outputs 9-11, 13-17
Read	VIL	VIL	5	5	DOUT
Program	Pulsed VIL to VIH	VIH	25	5	DIN

### Pin Names

A0-A10	Address Inputs
O <sub>0</sub> -O <sub>7</sub>	Data Outputs
CE/PGM	Chip Enable/Program
OE	Output Enable
VPP	Read 5V, Program 25V
VCC	5V
VSS	Ground



**Absolute Maximum Ratings** (Note 1)

Temperature Under Bias	-25°C to +85°C	Output Voltages with Respect to VSS	VCC + 0.3V to VSS - 0.3V
Storage Temperature	-65°C to +125°C	Lead Temperature (Soldering, 10 seconds)	300°C
VPP Supply Voltage with Respect to VSS	26.5V to -0.3V		
Input Voltages with Respect to VSS (except VPP) (Note 4)	VCC + 1 to -0.3V		

**READ OPERATION** (Note 2)**DC Operating Characteristics** TA = 0°C to +70°C, VCC = 5V ± 5%, VSS = 0V, unless otherwise noted.

Symbol	Parameter	Conditions	Min	Typ (Note 2)	Max	Units
ILI	Input Current	VIN = VCC or GND			10	μA
ILO	Output Leakage Current	VOUT = VCC or VSS (GND) CE/PGM = VIH			10	μA
VIL	Input Low Voltage		-0.1		0.8	V
VIH	Input High Voltage	(Note 4)	2.2		VCC + 1	V
VOL1	Output Low Voltage	IOL = 2.1 mA			0.45	V
VOH1	Output High Voltage	IOH = -400 μA	2.4			V
VOL2	Output Low Voltage	IOL = 0 μA			0.1	V
VOH2	Output High Voltage	IOH = 0 μA	VCC - 0.1			V
IPP1	VPP Supply Current	VPP = 5.25V			10	μA
ICC1	VCC Supply Current Active (TTL Levels)	CE/PGM, OE = VIL (Note 5) Addresses = VIH or VIL Frequency 1 MHz, I/O = 0 mA		2	10	mA
ICC2	VCC Supply Current Active (CMOS Levels)	CE/PGM, OE = VIL (Note 5) Addresses = GND or VCC Frequency 1 MHz, I/O = 0 mA		1	5	mA
ICCSB1	VCC Supply Current Standby	CE/PGM = VIH (Note 5)		0.1	1	mA
ICCSB2	VCC Supply Current Standby	CE/PGM = VCC (Note 5)			100	μA

**Capacitance** (Note 3) TA = 25°C, f = 1 MHz

Symbol	Parameter	Conditions	Typ	Max	Units
CI	Input Capacitance	VIN = 0V	4	6	pF
CO	Output Capacitance	VOUT = 0V	8	12	pF

**AC Test Conditions**

Input Pulse Levels	0.8V to 2.2V
Input Rise and Fall Times	20 ns
Timing	
Inputs	1V and 2V
Outputs	0.8V and 2V
Reference Levels	1.5V
Output Load	1 TTL Gate and CL = 100 pF

**Note 1:** "Absolute Maximum Ratings" are those values beyond which the safety of the device cannot be guaranteed. The functional operation of the device at these or any other conditions beyond those indicated in the "DC/AC Operating Characteristics" tables is not implied. Exposure to the absolute maximum rated conditions for extended periods may affect device reliability.

**Note 2:** Typical conditions are for operation at: TA = 25°C, VCC = 5V, VPP = VCC, and VSS = 0V.

**Note 3:** Capacitance is guaranteed by periodic testing. TA = 25°C, f = 1 MHz.

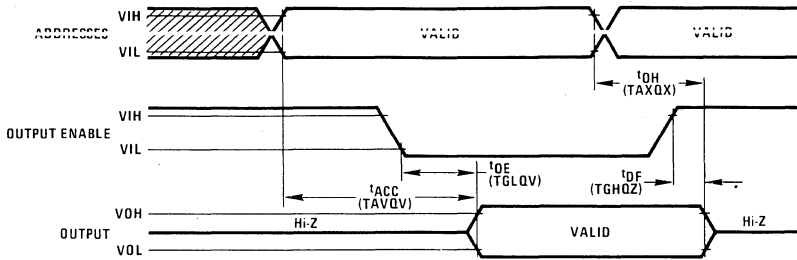
**Note 4:** The inputs (Address, OE, CE) may go above VCC by one volt with no latch up danger. Only the output (data inputs during programming) need be restricted to VCC + 0.3V to VSS - 0.3V.

## AC Characteristics $T_A = 0^\circ\text{C}$ to $+70^\circ\text{C}$ , $V_{CC} = 5\text{V} \pm 5\%$ , $V_{SS} = 0\text{V}$ , unless otherwise noted.

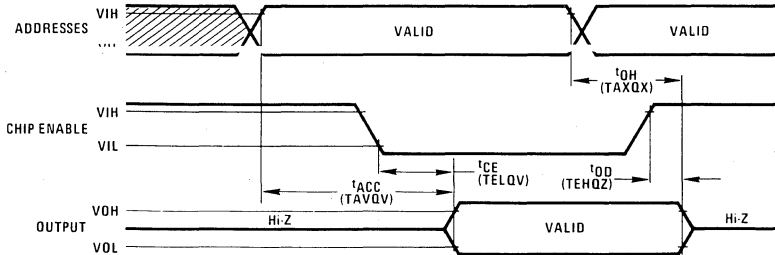
Symbol		Parameter	Conditions	NMC27C16-45		NMC27C16-55		NMC27C16-65		Units
Alternate	Standard			Min	Max	Min	Max	Min	Max	
$t_{ACC}$	TAVQV	Address to Output Delay	$\overline{CE}/PGM = \overline{OE} = \text{VIL}$		450		550		650	ns
$t_{CE}$	TELOV	$\overline{CE}$ to Output Delay	$\overline{OE} = \text{VIL}$		450		550		650	ns
$t_{OE}$	TGLQV	Output Enable to Output Valid	$\overline{CE}/PGM = \text{VIL}$		120		120		120	ns
$t_{DF}$	TGHQZ	Output Enable High to Output Hi-Z	$\overline{CE}/PGM = \text{VIL}$	0	100	0	100	0	100	ns
$t_{OH}$	TAXQX	Address to Output Hold	$\overline{CE}/PGM = \overline{OE} = \text{VIL}$	0		0		0		ns
$t_{OD}$	TEHQZ	$\overline{CE}$ to Output Hi-Z	$\overline{OE} = \text{VIL}$	0	100	0	100	0	100	ns

## Switching Time Waveforms

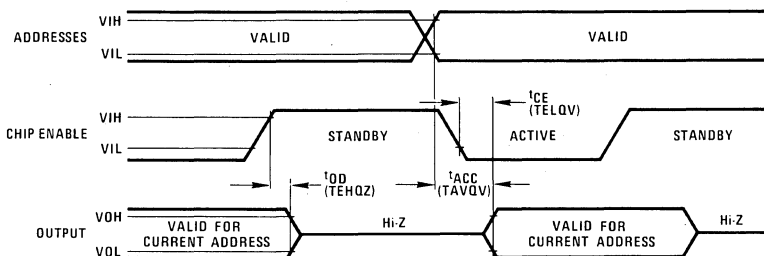
Read Cycle ( $\overline{CE}/PGM = \text{VIL}$ )



Read Cycle ( $\overline{OE} = \text{VIL}$ )



Standby Power-Down Mode ( $\overline{OE} = \text{VIL}$ )



## PROGRAM OPERATION

### DC Electrical Characteristics and Operating Conditions (Notes 5 and 6)

(TA = 25°C ± 5°C) (VCC = 5V ± 5%, VPP = 25V ± 0.5V)

Symbol	Parameter	Min	Typ	Max	Units
ILI	Input Leakage Current			10	μA
VIL	Input Low Level	- 0.1		0.8	V
VIH	Input High Level (Note 4)	2.2		VCC + 1	V
ICC	VCC Power Supply Current		2	10	mA
IPP1	VPP Supply Current (Note 7)			10	μA
IPP2	VPP Supply Current During Programming Pulse (Note 6)			30	mA

### AC Characteristics and Operating Conditions (Notes 1 and 2)

(TA = 25°C ± 5°C) (VCC = 5V ± 5%, VPP = 25V ± 1.0V)

Symbol	Parameter	Min	Typ	Max	Units
tAS	Address Set-up Time	2			μS
tOS	$\overline{OE}$ Set-up Time	2			μS
tDS	Data Set-up Time	2			μS
tAH	Address Hold Time	2			μS
tOH	$\overline{OE}$ Hold Time	2			μS
tDH	Data Hold Time	2			μS
tDF	Output Disable to Output TRI-STATE Delay (Note 7)	0		100	ns
tOE	Output Enable to Output Delay (Note 7)			120	ns
tPW	Program Pulse Width	45	50	55	ms
tPR	Program Pulse Rise Time	5			ns
tPF	Program Pulse Fall Time	5			ns
tVS	VPP Set-Up Time	2			μS
tVH	VPP Hold Time	2			μS

**Note 5:** VCC must be applied at the same time or before VPP and removed after or at the same time as VPP. To prevent damage to the device it must not be inserted into a board with power applied.

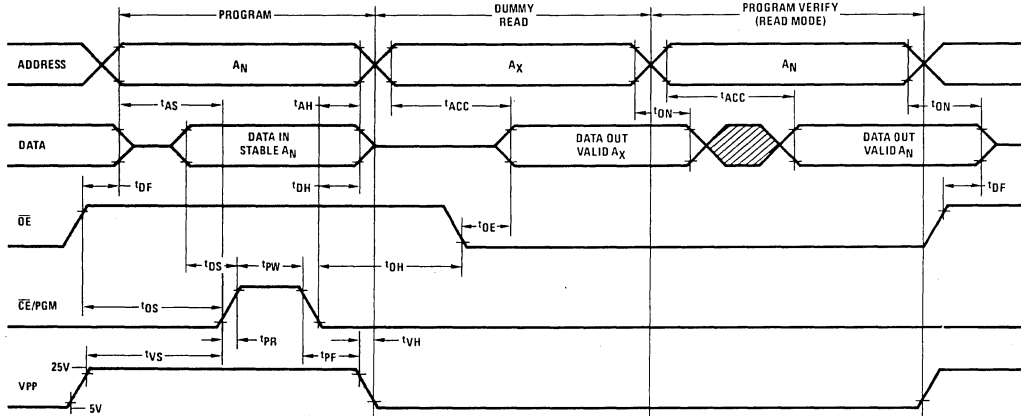
**Note 6:** Care must be taken to prevent overshoot of the VPP supply when switching to under 26V max.

**Note 7:**  $\overline{CE}/PGM = VIL$ , VPP = VCC.

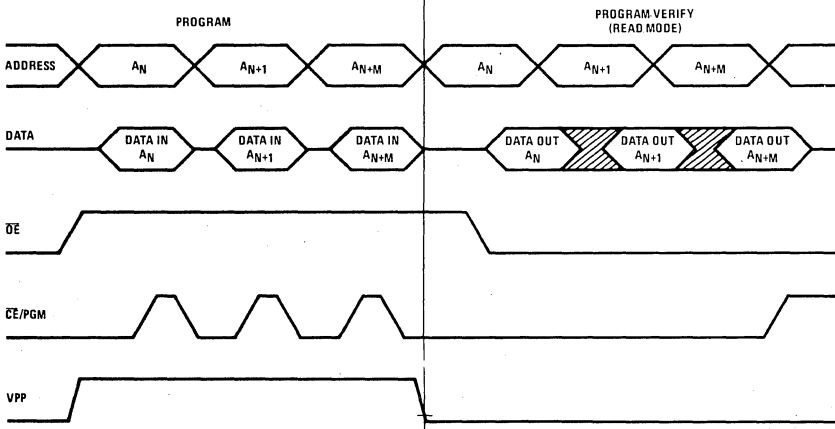
**Note 8:** The input timing reference level is 1V for VIL and 2V for VIH.

# PROGRAM Timing Diagrams

## Single Address Programming Followed by a Verify Mode



## Multiple Address Programming Followed by a Verify Mode\*



\* All timings are the same as the single address programming mode. A dummy read is required only if the last programmed byte is the first byte to be verified.

## Functional Description

### DEVICE OPERATION

The NMC27C16 has 3 modes of operation in the normal system environment. These are shown in Table I.

### Read Mode

The NMC27C16 read operation requires that  $\overline{OE} = V_{IL}$ ,  $\overline{CE}/PGM = V_{IL}$  and that addresses  $A_0-A_{10}$  have been stabilized. Valid data will appear on the output pins after  $t_{ACC}$ ,  $t_{OE}$  or  $t_{CE}$  times (see Switching Time Waveforms) depending on which is limiting.

TABLE I. OPERATING MODES (VCC = 5V)

Mode	Pin Name/Number		
	$\overline{CE}/PGM$ 18	$\overline{OE}$ 20	Outputs 9-11, 13-17
Read	VIL	VIL	DOUT
Deselect	Don't Care	VIH	Hi-Z
Standby	VIH	Don't Care	Hi-Z

### Deselect Mode

The NMC27C16 is deselected by making  $\overline{OE} = V_{IH}$ . This mode is independent of  $\overline{CE}/PGM$  and the condition of the addresses. The outputs are Hi-Z when  $\overline{OE} = V_{IH}$ . This allows OR-tying 2 or more NMC27C16s for memory expansion.

### Standby Mode (Power Down)

The NMC27C16 may be powered down to the standby mode by making  $\overline{CE}/PGM = V_{IH}$ . This is independent of  $\overline{OE}$  and automatically puts the outputs in their Hi-Z state. The power is reduced to 0.4% of the normal operating power. VCC must be maintained at 5V. Access time at power up remains either  $t_{ACC}$  or  $t_{CE}$  (see Switching Time Waveforms).

### PROGRAMMING

The NMC27C16 is shipped from National completely erased. All bits will be at a "1" level (output high) in this initial state and after any full erasure. Table II shows the 3 programming modes.

## Functional Description (Continued)

TABLE II. PROGRAMMING MODES (VCC = 5V)

Mode	Pin Name/Number			
	$\overline{CE}/PGM$ 18	$\overline{OE}$ 20	VPP 21	Outputs Q 9-11, 13-17
Program	Pulsed VIL to VIH	VIH	25	DIN
Program Verify	VIL	VIL	5	DOUT
Program Inhibit	VIL	VIH	25	Hi-Z

### Program Mode

The NMC27C16 is programmed by introducing "0"s into the desired locations. This is done 8 bits (a byte) at a time. Any individual address, a sequence of addresses, or addresses chosen at random may be programmed. Any or all of the 8 bits associated with an address location may be programmed with a single program pulse applied to the chip enable pin. All input voltage levels, including the program pulse on chip enable are TTL compatible. The programming sequence is:

With  $VPP = 25V$ ,  $VCC = 5V$ ,  $\overline{OE} = VIH$  and  $\overline{CE}/PGM = VIL$ , an address is selected and the desired data word is applied to the output pins. (VIL = "0" and VIH = "1" for both address and data.) After the address and data signals are stable the program pin is pulsed from VIL to VIH with a pulse width between 45 ms and 55 ms.

Multiple pulses are not needed but will not cause device damage. No pins should be left open. A high level (VIH or higher) *must not* be maintained longer than  $t_{PW(MAX)}$  on the program pin during programming. NMC27C16s may be programmed in parallel with the same data in this mode.

### Program Verify Mode

The programming of the NMC27C16 is verified in the program verify mode which has VPP at VCC (see Table II). **After programming an address, that same address cannot be immediately verified without an address change (dummy read).**

### Program Inhibit Mode

The program inhibit mode allows programming several NMC27C16s simultaneously with different data for each one by controlling which ones receive the program pulse. All similar inputs of the NMC27C16 may be paralleled. Pulsing the program pin (from VIL to VIH) will program a unit while inhibiting the program pulse to a unit will keep it from being programmed and keeping  $\overline{OE} = VIH$  will put its outputs in the Hi-Z state.

### ERASING

The NMC27C16 is erased by exposure to high intensity ultraviolet light through the transparent window. This exposure discharges the floating gate to its initial state through induced photo current. It is recommended that the NMC27C16 be kept out of direct sunlight. The UV content of sunlight may cause a partial erasure of some bits in a relatively short period of time. Direct sunlight can also cause temporary functional failure. Extended exposure to room level fluorescent lighting will also cause erasure. An opaque coating (paint, tape, label, etc.) should be placed over the package window if this product is to be operated under these lighting conditions. Covering the window also reduces ICC due to photodiode currents.

An ultraviolet source of 2537Å yielding a total integrated dosage of 15 watt-seconds/cm<sup>2</sup> is required. This will erase the part in approximately 15 to 20 minutes if a UV lamp with a 12,000 μW/cm<sup>2</sup> power rating is used. The NMC27C16 to be erased should be placed 1 inch away from the lamp and no filters should be used.

An erasure system should be calibrated periodically. The distance from lamp to unit should be maintained at 1 inch. The erasure time is increased by the square of the distance (if the distance is doubled the erasure time goes up by a factor of 4). Lamps lose intensity as they age. When a lamp is changed, the distance is changed, or the lamp is aged, the system should be checked to make certain full erasure is occurring. Incomplete erasure will cause symptoms that can be misleading. Programmers, components, and system designs have been erroneously suspected when incomplete erasure was the basic problem.

## MM2758 8192-Bit (1024 × 8) UV Erasable PROM

### General Description

The MM2758 is a high speed 8k UV erasable and electrically reprogrammable EPROM ideally suited for applications where fast turn-around and pattern experimentation are important requirements.

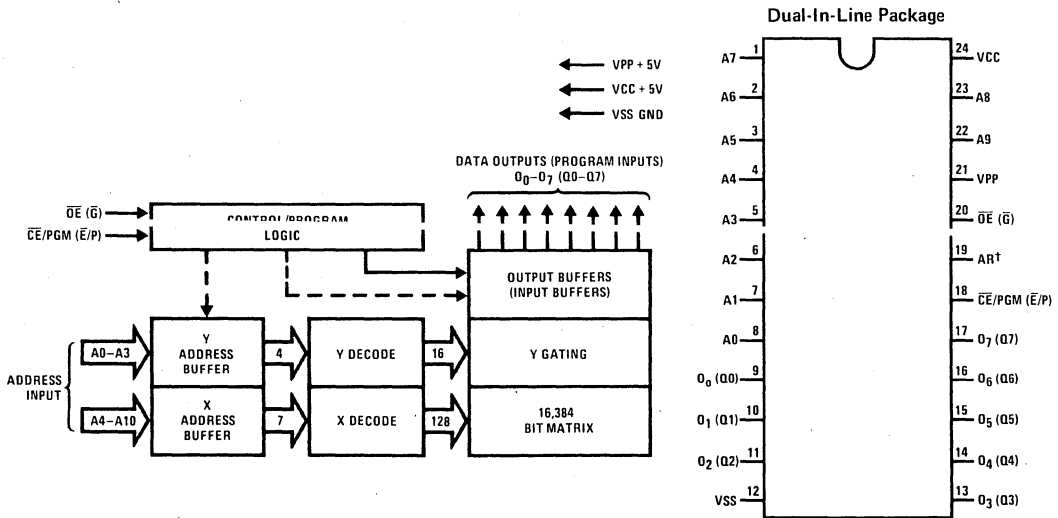
The MM2758 is packaged in a 24-pin dual-in-line package with transparent lid. The transparent lid allows the user to expose the chip to ultraviolet light to erase the bit pattern. A new pattern can then be written into the device by following the programming procedure.

This EPROM is fabricated with the reliable, high volume, time proven, N-channel silicon gate technology.

### Features

- 1024 × 8 organization
- 525 mW max active power, 132 mW max standby power
- Low power during programming
- Access time—450 ns
- Single 5V power supply
- Static—no clocks required
- Inputs and outputs TTL compatible during both read and program modes
- TRI-STATE® output

### Block and Connection Diagrams \*



Pin Connection During Read or Program

MODE	PIN NAME/NUMBER				
	CE/PGM (E/P) 18	OE (G) 20	VPP 21	VCC 24	OUTPUTS 9-11, 13-17
Read	VIL	VIL	5	5	DOUT
Program	Pulsed VIL to VIH	VIH	25	5	DIN

\*Symbols in parentheses are proposed industry standard

†For MM2758A AR = VIL for all operating modes

For MM2758B AR = VIH for all operating modes

TOP VIEW  
Order Number MM2758AQ  
or MM2758BQ  
See NS Package J24CQ

#### Pin Names

A0-A10	Address Inputs
O <sub>0</sub> -O <sub>7</sub> (Q <sub>0</sub> -Q <sub>7</sub> )	Data Outputs
CE/PGM (E/P)	Chip Enable/Program
OE (G)	Output Enable
VPP	Read 5V, Program 25V
VCC	Power (5V)
VSS	Ground

## Absolute Maximum Ratings (Note 1)

Temperature Under Bias	-25°C to +85°C	All Input or Output Voltages with Respect to VSS (except VPP)	6V to -0.3V
Storage Temperature	-65°C to +125°C	Power Dissipation	1.5 W
VPP Supply Voltage with Respect to VSS	26.5V to -0.3V	Lead Temperature (Soldering, 10 seconds)	300°C

## READ OPERATION (Note 2)

### DC Operating Characteristics

$T_A = 0^\circ\text{C}$  to  $+70^\circ\text{C}$ ,  $V_{CC} = 5V \pm 5\%$ ,  
 $V_{PP} = V_{CC} \pm 0.6V$  (Note 3),  $V_{SS} = 0V$ , unless otherwise noted.

SYMBOL	PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
ILI	Input Current	$V_{IN} = 5.25V$ or $V_{IN} = V_{IL}$			10	$\mu\text{A}$
ILO	Output Leakage Current	$V_{OUT} = 5.25V$ , $\overline{CE}/\overline{PGM} = 5V$			10	$\mu\text{A}$
IPP1	VPP Supply Current	$V_{PP} = 5.85V$			5	mA
ICC1	VCC Supply Current (Standby)	$\overline{CE}/\overline{PGM} = V_{IH}$ , $\overline{OE} = V_{IL}$		10	25	mA
ICC2	VCC Supply Current (Active)	$\overline{CE}/\overline{PGM} = \overline{OE} = V_{IL}$		57	100	mA
VIL	Input Low Voltage		0.1		0.8	V
VIH	Input High Voltage		2.0		$V_{CC} \cdot 1$	V
VOH	Output High Voltage	$I_{OH} = 400 \mu\text{A}$	2.4			V
VOL	Output Low Voltage	$I_{OL} = 2.1 \text{ mA}$			0.45	V

## AC Characteristics (Note 4)

$T_A = 0^\circ\text{C}$  to  $+70^\circ\text{C}$ ,  $V_{CC} = 5V \pm 5\%$ ,  
 $V_{PP} = V_{CC} \pm 0.6V$  (Note 3),  $V_{SS} = 0V$ , unless otherwise noted.

SYMBOL		PARAMETER	CONDITIONS	MM2758		UNITS
ALTERNATE	STANDARD			MIN	MAX	
t <sub>ACC</sub>	TAVQV	Address to Output Delay	$\overline{CE}/\overline{PGM} = \overline{OE} = V_{IL}$		450	ns
t <sub>CE</sub>	TELQV	$\overline{CE}$ to Output Delay	$\overline{OE} = V_{IL}$		450	ns
t <sub>OE</sub>	TGLOV	Output Enable to Output Delay	$\overline{CE}/\overline{PGM} = V_{IL}$		120	ns
t <sub>DF</sub>	TGHQZ	Output Enable High to Output Hi-Z	$\overline{CE}/\overline{PGM} = V_{IL}$	0	100	ns
t <sub>OH</sub>	TAXQX	Address to Output Hold	$\overline{CE}/\overline{PGM} = \overline{OE} = V_{IL}$	0		ns
t <sub>OD</sub>	TEHQZ	$\overline{CE}$ to Output Hi-Z	$\overline{OE} = V_{IL}$	0	100	ns

## Capacitance (Note 5)

$T_A = 25^\circ\text{C}$ ,  $f = 1 \text{ MHz}$

SYMBOL	PARAMETER	CONDITIONS	TYP	MAX	UNITS
CI	Input Capacitance	$V_{IN} = 0V$	4	6	pF
CO	Output Capacitance	$V_{OUT} = 0V$	8	12	pF

**Note 1:** "Absolute Maximum Ratings" are those values beyond which the safety of the device cannot be guaranteed. Except for "Operating Temperature Range" they are not meant to imply that the devices should be operated at these limits. The table of "Electrical Characteristics" provides conditions for actual device operation.

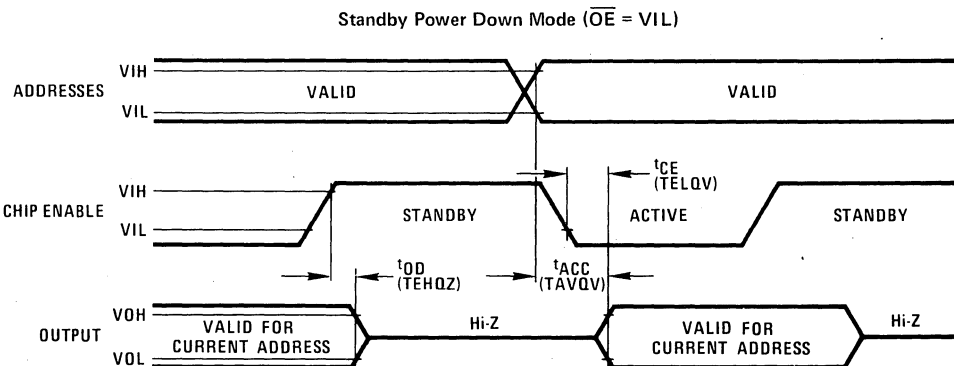
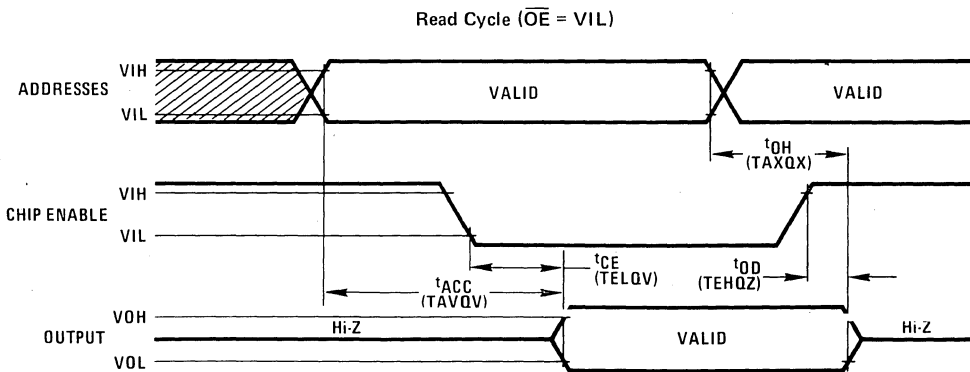
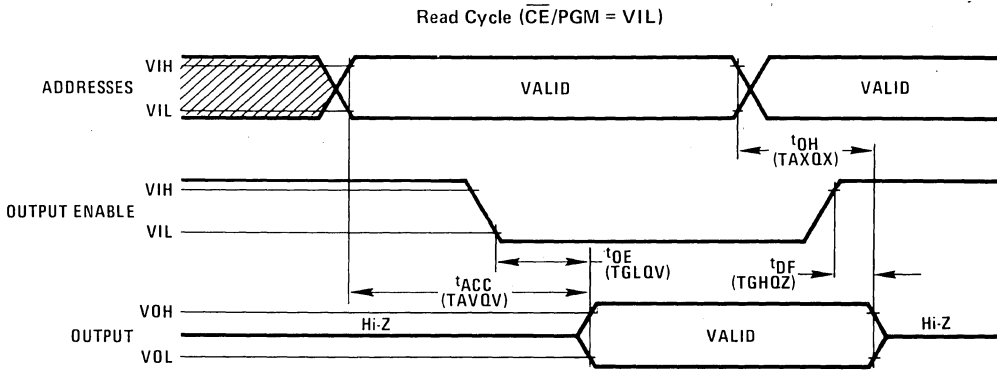
**Note 2:** Typical conditions are for operation at:  $T_A = 25^\circ\text{C}$ ,  $V_{CC} = 5V$ ,  $V_{PP} = V_{CC}$ , and  $V_{SS} = 0V$ .

**Note 3:** VPP may be connected to VCC except during program. The  $\pm 0.6V$  tolerance allows a circuit to switch VPP between the read voltage and the program voltage.

**Note 4:** Output load: 1 TTL gate and  $CL = 100 \text{ pF}$ . Input rise and fall times  $\leq 20 \text{ ns}$ .

**Note 5:** Capacitance is guaranteed by periodic testing.

# Switching Time Waveforms \*



\*Symbols in parentheses are proposed industry standard



## PROGRAM OPERATION

### DC Electrical Characteristics and Operating Conditions (Notes 1 and 2)

( $T_A = 25^\circ\text{C} \pm 5^\circ\text{C}$ ) ( $V_{CC} = 5\text{V} \pm 5\%$ ,  $V_{PP} = 25\text{V} \pm 1\text{V}$ )

SYMBOL	PARAMETER	MIN	TYP	MAX	UNITS
ILI	Input Leakage Current (Note 3)			10	$\mu\text{A}$
VIL	Input Low Level	-0.1		0.8	V
VIH	Input High Level	2.0		$V_{CC} + 1$	V
ICC	VCC Power Supply Current			100	mA
IPP1	VPP Supply Current (Note 4)			5	mA
IPP2	VPP Supply Current During Programming Pulse (Note 5)			30	mA

### AC Characteristics and Operating Conditions (Notes 1, 2, and 6)

( $T_A = 25^\circ\text{C} \pm 5^\circ\text{C}$ ) ( $V_{CC} = 5\text{V} \pm 5\%$ ,  $V_{PP} = 25\text{V} \pm 1\text{V}$ )

SYMBOL		PARAMETER	MIN	TYP	MAX	UNITS
ALTERNATE	STANDARD					
tAS	TAVPH	Address Setup Time	2			$\mu\text{s}$
tOS	TGHPH	$\overline{\text{OE}}$ Setup Time	2			$\mu\text{s}$
tDS	TDVPH	Data Setup Time	2			$\mu\text{s}$
tAH	TPLAX	Address Hold Time	2			$\mu\text{s}$
tOH	TPLGX	$\overline{\text{OE}}$ Hold Time	2			$\mu\text{s}$
tDH	TPLDX	Data Hold Time	2			$\mu\text{s}$
tDF	TGHQZ	Chip Disable to Output Float Delay (Note 4)	0		100	ns
tCE	TGLOV	Chip Enable to Output Delay (Note 4)			120	ns
tpW	TPHPL	Program Pulse Width	45	50	55	ms
tpR	TPH1PH2	Program Pulse Rise Time	5			ns
tpF	TPL2PL1	Program Pulse Fall Time	5			ns

**Note 1:** VCC must be applied at the same time or before VPP and removed after or at the same time as VPP. To prevent damage to the device it must not be inserted into a board with power applied.

**Note 2:** Care must be taken to prevent overshoot of the VPP supply when switching to +25V.

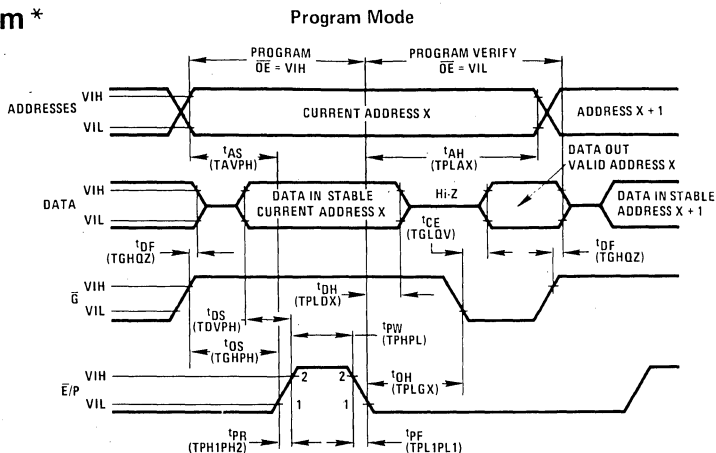
**Note 3:**  $0.45\text{V} \leq V_{IN} < 5.25\text{V}$ .

**Note 4:**  $\overline{\text{CE}}/\text{PGM} = V_{IL}$ ,  $V_{PP} = V_{CC} + 0.6\text{V}$ .

**Note 5:**  $V_{PP} = 26\text{V}$ .

**Note 6:** Transition times  $\leq 20$  ns unless noted otherwise.

## Timing Diagram \*



## Functional Description

## DEVICE OPERATION

The MM2758 has 3 modes of operation in the normal system environment. These are shown in Table I.

## Read Mode

The MM2758 read operation requires that  $\overline{OE} = VIL$ ,  $\overline{CE}/PGM = VIL$  and that addresses A0–A10 have been stabilized. Valid data will appear on the output pins after  $t_{ACC}$ ,  $t_{OE}$  or  $t_{CE}$  times (see Switching Time Waveforms) depending on which is limiting.

## Deselect Mode

The MM2758 is deselected by making  $\overline{OE} = VIH$ . This mode is independent of  $\overline{CE}/PGM$  and the condition of the addresses. The outputs are Hi-Z when  $\overline{OE} = VIH$ . This allows OR-tying 2 or more MM2716's for memory expansion.

## Standby Mode (Power Down)

The MM2758 may be powered down to the standby mode by making  $\overline{CE}/PGM = VIH$ . This is independent of  $\overline{OE}$  and automatically puts the outputs in their Hi-Z state. The power is reduced to 25% (132 mW max) of the normal operating power. VCC and VPP must be maintained at 5V. Access time at power up remains either  $t_{ACC}$  or  $t_{CE}$  (see Switching Time Waveforms).

## PROGRAMMING

The MM2758 is shipped from National completely erased. All bits will be at a "1" level (output high) in this initial state and after any full erasure. Table II shows the 3 programming modes.

TABLE I. OPERATING MODES (VCC = VPP = 5V)

MODE	PIN NAME/NUMBER		
	$\overline{CE}/PGM$ ( $\overline{E}/P$ ) 18	$\overline{OE}$ ( $\overline{G}$ ) 20	OUTPUTS 9–11, 13–17
Read	VIL	VIL	DOUT
Deselect	Don't Care	VIH	Hi-Z
Standby	VIH	Don't Care	Hi-Z

TABLE II. PROGRAMMING MODES (VCC = 5V)

MODE	PIN NAME/NUMBER			
	$\overline{CE}/PGM$ ( $\overline{E}/P$ ) 18	$\overline{OE}$ ( $\overline{G}$ ) 20	VPP 21	OUTPUTS Q 9–11, 13–17
Program	Pulsed VIL to VIH	VIH	25	DIN
Program Verify	VIL	VIL	25(5)	DOUT
Program Inhibit	VIL	VIH	25	Hi-Z

\*Symbols in parentheses are proposed industry standard

## Functional Description (Continued)

### Program Mode

The MM2758 is programmed by introducing "0"s into the desired locations. This is done 8 bits (a byte) at a time. Any individual address, a sequence of addresses, or addresses chosen at random may be programmed. Any or all of the 8 bits associated with an address location may be programmed with a single program pulse applied to the chip enable pin. All input voltage levels, including the program pulse on chip enable are TTL compatible. The programming sequence is:

With  $V_{PP} = 25V$ ,  $V_{CC} = 5V$ ,  $\overline{OE} = V_{IH}$  and  $\overline{CE}/PGM = V_{IL}$ , an address is selected and the desired data word is applied to the output pins. ( $V_{IL} = "0"$  and  $V_{IH} = "1"$  for both address and data.) After the address and data signals are stable the program pin is pulsed from  $V_{IL}$  to  $V_{IH}$  with a pulse width between 45 ms and 55 ms.

Multiple pulses are not needed but will not cause device damage. No pins should be left open. A high level ( $V_{IH}$  or higher) *must not* be maintained longer than  $tpw(MAX)$  on the program pin during programming. MM2758's may be programmed in parallel with the same data in this mode.

### Program Verify Mode

The programming of the MM2758 may be verified either 1 word at a time during the programming (as shown in the timing diagram) or by reading all of the words out at the end of the programming sequence. This can be done with  $V_{PP} = 25V$  (or 5V) in either case.

### Program Inhibit Mode

The program inhibit mode allows programming several MM2758s simultaneously with different data for each one by controlling which ones receive the program pulse. All similar inputs of the MM2758 may be paralleled. Pulsing the program pin (from  $V_{IL}$  to  $V_{IH}$ ) will program

a unit while inhibiting the program pulse to a unit will keep it from being programmed and keeping  $\overline{OE} = V_{IH}$  will put its outputs in the Hi-Z state.

### ERASING

The MM2758 is erased by exposure to high intensity ultraviolet light through the transparent window. This exposure discharges the floating gate to its initial state through induced photo current. It is recommended that the MM2758 be kept out of direct sunlight. The UV content of sunlight may cause a partial erasure of some bits in a relatively short period of time. Direct sunlight can also cause temporary functional failure. Extended exposure to room level fluorescent lighting will also cause erasure. An opaque coating (paint, tape, label, etc.) should be placed over the package window if this product is used under these lighting conditions.

An ultraviolet source of 2537 Å yielding a total integrated dosage of 15 watt-seconds/cm<sup>2</sup> is required. This will erase the part in approximately 15 to 20 minutes if a UV lamp with a 12,000 μW/cm<sup>2</sup> power rating is used. The MM2758 to be erased should be placed 1 inch away from the lamp and no filters should be used.

An erasure system should be calibrated periodically. The distance from lamp to unit should be maintained at 1 inch. The erasure time is increased by the square of the distance (if the distance is doubled the erasure time goes up by a factor of 4). Lamps lose intensity as they age. When a lamp is changed, the distance is changed, or the lamp is aged, the system should be checked to make certain full erasure is occurring. Incomplete erasure will cause symptoms that can be misleading. Programmers, components, and system designs have been erroneously suspected when incomplete erasure was the basic problem.

# MM54C373/MM74C373 TRI-STATE® Octal D-Type Latch

# MM54C374/MM74C374 TRI-STATE® Octal D-Type Flip-Flop

## General Description

The MM54C373/MM74C373, MM54C374/MM74C374 are integrated, complementary MOS (CMOS), 8-bit storage elements with TRI-STATE® outputs. These outputs have been specially designed to drive highly capacitive loads, such as one might find when driving a bus, and to have a fan-out of 1 when driving standard TTL. When a high logic level is applied to the OUTPUT DISABLE input, all outputs go to a high impedance state, regardless of what signals are present at the other inputs and the state of the storage elements.

The MM54C373/MM74C373 is an 8-bit latch. When LATCH ENABLE is high the Q outputs will follow the D inputs. When LATCH ENABLE goes low, data at the D inputs, which meets the set-up and hold time requirements, will be retained at the outputs until LATCH ENABLE returns high again.

The MM54C374/MM74C374 is an 8-bit, D-type, positive-edge triggered flip-flop. Data at the D inputs, meeting

the set-up and hold time requirements, is transferred to the Q outputs on positive-going transitions of the CLOCK input.

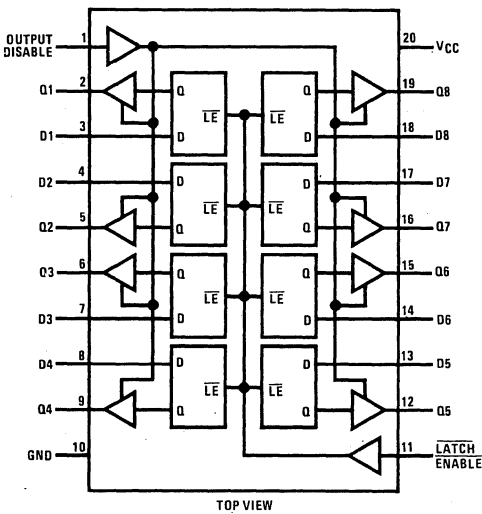
Both the MM54C373/MM74C373 and the MM54C374/MM74C374 are being assembled in 20-pin dual-in-line packages with 0.300" pin centers.

## Features

- Wide supply voltage range 3.0V to 15V
- High noise immunity 0.45 V<sub>CC</sub> typ
- Low power consumption
- TTL compatibility fan-out of 1 driving standard TTL
- Bus driving capability
- TRI-STATE outputs
- Eight storage elements in one package
- Single CLOCK/LATCH ENABLE and OUTPUT DISABLE control inputs
- 20-pin dual-in-line package with 0.300" centers takes half the board space of a 24-pin package

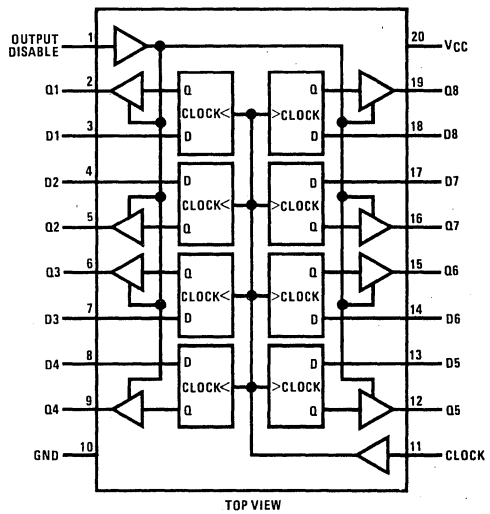
## Connection Diagrams

Dual-In-Line Package



Order Number MM54C373J or MM74C373N  
See NS Package J20A or N20A

Dual-In-Line Package



Order Number MM54C374J or MM74C374N  
See NS Package J20A or N20A

**Absolute Maximum Ratings** (Note 1)

Voltage at Any Pin	-0.3V to $V_{CC} + 0.3V$	Package Dissipation	500 mW
Operating Temperature Range		Operating $V_{CC}$ Range	3V to 15V
MM54C373, MM54C374	-55°C to +125°C	Absolute Maximum $V_{CC}$	18V
MM74C373, MM74C374	-40°C to +85°C	Lead Temperature (Soldering, 10 seconds)	300°C
Storage Temperature Range	-65°C to +150°C		

**Electrical Characteristics** Min/max limits apply across temperature range, unless otherwise noted.

PARAMETER		CONDITIONS	MIN	TYP	MAX	UNITS
<b>CMOS TO CMOS</b>						
$V_{IN(1)}$	Logical "1" Input Voltage	$V_{CC} = 5V$ $V_{CC} = 10V$	3.5 8.0			V V
$V_{IN(0)}$	Logical "0" Input Voltage	$V_{CC} = 5V$ $V_{CC} = 10V$			1.5 2.0	V V
$V_{OUT(1)}$	Logical "1" Output Voltage	$V_{CC} = 5V, I_O = -10 \mu A$ $V_{CC} = 10V, I_O = -10 \mu A$	4.5 9.0			V V
$V_{OUT(0)}$	Logical "0" Output Voltage	$V_{CC} = 5V, I_O = 10 \mu A$ $V_{CC} = 10V, I_O = 10 \mu A$			0.5 1.0	V V
$I_{IN(1)}$	Logical "1" Input Current	$V_{CC} = 15V, V_{IN} = 15V$		0.005	1.0	$\mu A$
$I_{IN(0)}$	Logical "0" Input Current	$V_{CC} = 15V, V_{IN} = 0V$	-1.0	-0.005		$\mu A$
$I_{OZ}$	TRI-STATE Leakage Current	$V_{CC} = 15V, V_O = 15V$ $V_{CC} = 15V, V_O = 0V$		0.005 -0.005	1.0	$\mu A$ $\mu A$
$I_{CC}$	Supply Current	$V_{CC} = 15V$		0.05	300	$\mu A$
<b>CMOS/LPTTL INTERFACE</b>						
$V_{IN(1)}$	Logical "1" Input Voltage	54C, $V_{CC} = 4.5V$ 74C, $V_{CC} = 4.75V$	$V_{CC}-1.5$ $V_{CC}-1.5$			V V
$V_{IN(0)}$	Logical "0" Input Voltage	54C, $V_{CC} = 4.5V$ 74C, $V_{CC} = 4.75V$			0.8 0.8	V V
$V_{OUT(1)}$	Logical "1" Output Voltage	54C, $V_{CC} = 4.5V, I_O = -360 \mu A$ 74C, $V_{CC} = 4.75V, I_O = -360 \mu A$	$V_{CC}-0.4$ $V_{CC}-0.4$			V V
		54C, $V_{CC} = 4.5V, I_O = -1.6 mA$ 74C, $V_{CC} = 4.75V, I_O = -1.6 mA$	2.4 2.4			V V
$V_{OUT(0)}$	Logical "0" Output Voltage	54C, $V_{CC} = 4.5V, I_O = 1.6 mA$ 74C, $V_{CC} = 4.75V, I_O = 1.6 mA$			0.4 0.4	V V
<b>OUTPUT DRIVE</b>						
$I_{SOURCE}$	Output Source Current	$V_{CC} = 5V, V_{OUT} = 0V, T_A = 25^\circ C$ , (Note 4)	-12.0	-24		mA
$I_{SOURCE}$	Output Source Current	$V_{CC} = 10V, V_{OUT} = 0V, T_A = 25^\circ C$ , (Note 4)	-24.0	-48		mA
$I_{SINK}$	Output Sink Current (N-Channel)	$V_{CC} = 5V, V_{OUT} = V_{CC}, T_A = 25^\circ C$ , (Note 4)	6.0	12		mA
$I_{SINK}$	Output Sink Current (N-Channel)	$V_{CC} = 10V, V_{OUT} = V_{CC}, T_A = 25^\circ C$ , (Note 4)	24.0	48		mA
<b>Switching Characteristics</b> $T_A = 25^\circ C, C_L = 50 pF, t_r = t_f = 20 ns$ , unless otherwise specified.						
PARAMETER		CONDITIONS	MIN	TYP	MAX	UNITS
$t_{pd1}, t_{pd0}$	Propagation Delay, LATCH ENABLE to Output	$V_{CC} = 5V, C_L = 50 pF$ $V_{CC} = 10V, C_L = 50 pF$ $V_{CC} = 5V, C_L = 150 pF$ $V_{CC} = 10V, C_L = 150 pF$		165 70 195 85	330 140 390 170	ns ns ns ns

# Switching Characteristics (Continued) $T_A = 25^\circ\text{C}$ , $C_L = 50\text{ pF}$ , $t_r = t_f = 20\text{ ns}$ , unless otherwise specified.

MM54C373/MM74C373, MM54C374/MM74C374

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS
t <sub>pd1</sub> , t <sub>pd0</sub> Propagation Delay Data In to Output MM54C373, MM74C373	LATCH ENABLE = V <sub>CC</sub>				
	V <sub>CC</sub> = 5V, C <sub>L</sub> = 50 pF		155	310	ns
	V <sub>CC</sub> = 10V, C <sub>L</sub> = 50 pF		70	140	ns
	V <sub>CC</sub> = 5V, C <sub>L</sub> = 150 pF		185	370	ns
t <sub>pd1</sub> , t <sub>pd0</sub> Propagation Delay CLOCK to Output MM54C374/MM74C374	V <sub>CC</sub> = 5V, C <sub>L</sub> = 50 pF		150	300	ns
	V <sub>CC</sub> = 10V, C <sub>L</sub> = 50 pF		65	130	ns
	V <sub>CC</sub> = 5V, C <sub>L</sub> = 150 pF		180	360	ns
	V <sub>CC</sub> = 10V, C <sub>L</sub> = 150 pF		80	160	ns
t <sub>SET-UP</sub> Minimum Set-Up Time Data In to CLOCK/LATCH ENABLE	t <sub>HOLD</sub> = 0 ns				
	V <sub>CC</sub> = 5V		70	140	ns
	V <sub>CC</sub> = 10V		35	70	ns
t <sub>PWH</sub> Minimum LATCH ENABLE Pulse Width MM54C373, MM74C373	V <sub>CC</sub> = 5V		75	150	ns
	V <sub>CC</sub> = 10V		55	110	ns
t <sub>PWH</sub> , t <sub>PWL</sub> Minimum CLOCK Pulse Width MM54C374, MM74C374	V <sub>CC</sub> = 5V		70	140	ns
	V <sub>CC</sub> = 10V		50	100	ns
f <sub>MAX</sub> Maximum LATCH ENABLE Frequency MM54C373, MM74C373	V <sub>CC</sub> = 5V	3.3	6.7		MHz
	V <sub>CC</sub> = 10V	4.5	9.0		MHz
f <sub>MAX</sub> Maximum CLOCK Frequency MM54C374, MM74C374	V <sub>CC</sub> = 5V	3.5	7.0		MHz
	V <sub>CC</sub> = 10V	5.0	10.0		MHz
t <sub>1H</sub> , t <sub>0H</sub> Propagation Delay OUTPUT DISABLE to High Impedance State (From a Logic Level)	R <sub>L</sub> = 10k, C <sub>L</sub> = 5 pF				
	V <sub>CC</sub> = 5V		105	210	ns
	V <sub>CC</sub> = 10V		60	120	ns
t <sub>H1</sub> , t <sub>H0</sub> Propagation Delay OUTPUT DISABLE to Logic Level (From High Impedance State)	R <sub>L</sub> = 10k, C <sub>L</sub> = 50 pF				
	V <sub>CC</sub> = 5V		105	210	ns
	V <sub>CC</sub> = 10V		45	90	ns
t <sub>THL</sub> , t <sub>TLH</sub> Transition Time	V <sub>CC</sub> = 5V, C <sub>L</sub> = 50 pF		65	130	ns
	V <sub>CC</sub> = 10V, C <sub>L</sub> = 50 pF		35	70	ns
	V <sub>CC</sub> = 5V, C <sub>L</sub> = 150 pF		110	220	ns
	V <sub>CC</sub> = 10V, C <sub>L</sub> = 150 pF		70	140	ns
t <sub>r</sub> , t <sub>f</sub> Maximum LATCH ENABLE Rise and Fall Time MM54C373, MM74C373	V <sub>CC</sub> = 5V		NA		μs
	V <sub>CC</sub> = 10V		NA		μs
t <sub>r</sub> , t <sub>f</sub> Maximum CLOCK Rise and Fall Time MM54C374, MM74C374	V <sub>CC</sub> = 5V	15	>2000		μs
	V <sub>CC</sub> = 10V	5	>2000		μs
C <sub>CLK</sub> , C <sub>LE</sub> Input Capacitance	CLOCK/ $\overline{\text{LE}}$ Input, (Note 2)		7.5	10	pF
C <sub>OD</sub> Input Capacitance	OUTPUT DISABLE Input, (Note 2)		7.5	10	pF
C <sub>IN</sub> Input Capacitance	Any Other Input, (Note 2)		5.0	7.5	pF
C <sub>OUT</sub> Output Capacitance	High Impedance State, (Note 2)		10	15	pF
C <sub>PD</sub> Power Dissipation Capacitance MM54C373, MM74C373	Per Package, (Note 3)		200		pF
C <sub>PD</sub> Power Dissipation Capacitance MM54C374, MM74C374	Per Package, (Note 3)		250		pF

**Note 1:** "Absolute Maximum Ratings" are those values beyond which the safety of the device cannot be guaranteed. Except for "Operating Range" they are not meant to imply that the devices should be operated at these limits. The table of "Electrical Characteristics" provides conditions for actual device operation.

**Note 2:** Capacitance is guaranteed by periodic testing.

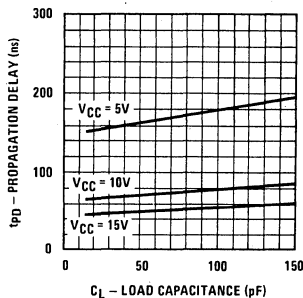
**Note 3:** C<sub>PD</sub> determines the no load ac power consumption of any CMOS device. For complete explanation see 54C/74C Family Characteristics application note, AN-90.

**Note 4:** These are peak output current capabilities. Continuous output current is rated at 12 mA max.

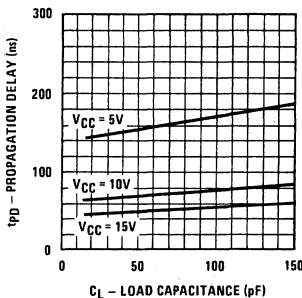
7

## Typical Performance Characteristics $T_A = 25^\circ\text{C}$

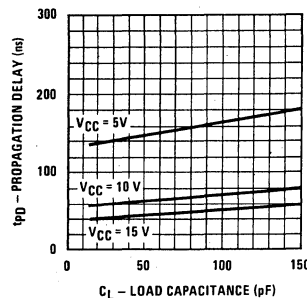
**MM54C373/MM74C373**  
Propagation Delay, LATCH  
ENABLE to Output vs Load  
Capacitance



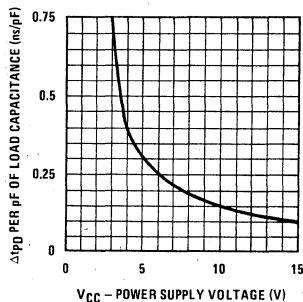
**MM54C373/MM74C373**  
Propagation Delay, Data In to Output  
vs Load Capacitance



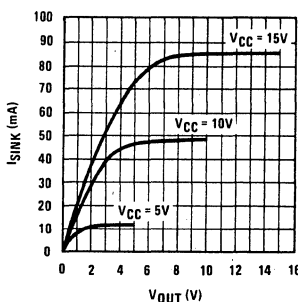
**MM54C374/MM74C374**  
Propagation Delay, CLOCK to Output  
vs Load Capacitance



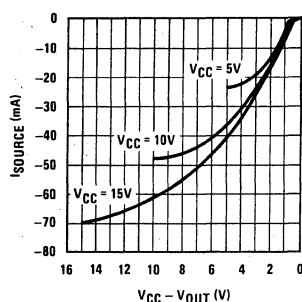
**MM54C373/MM74C373,**  
**MM54C374/MM74C374**  
Change in Propagation Delay per pF of  
Load Capacitance ( $\Delta t_{PD}/pF$ ) vs Power  
Supply Voltage



**MM54C373/MM74C373,**  
**MM54C374/MM74C374**  
Output Sink Current vs  $V_{OUT}$



**MM54C373/MM74C373,**  
**MM54C374/MM74C374** Output  
Source Current vs  $V_{CC} - V_{OUT}$



## Truth Tables

**MM54C373/MM74C373**

OUTPUT DISABLE	LATCH ENABLE	D	Q
L	H	H	H
L	H	L	L
L	L	X	Q
H	X	X	Hi-Z

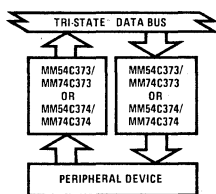
**MM54C374/MM74C374**

OUTPUT DISABLE	CLOCK	D	Q
L		H	H
L		L	L
L	L	X	Q
L	H	X	Q
H	X	X	Hi-Z

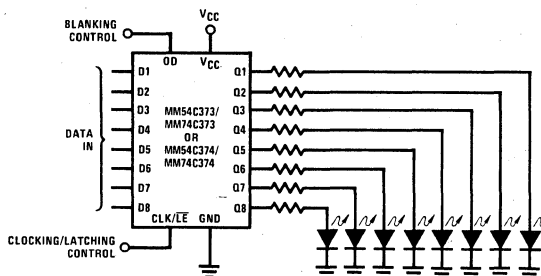
L = low logic level  
H = high logic level  
X = irrelevant  
 = low to high logic level transition  
Q = preexisting output level  
Hi-Z = high impedance output state

## Typical Applications

Data Bus Interfacing Element



Simple, Latching, Octal, LED Indicator Driver with Blanking  
For Use As Data Display, Bus Monitor,  
 $\mu\text{P}$  Front Panel Display, Etc.

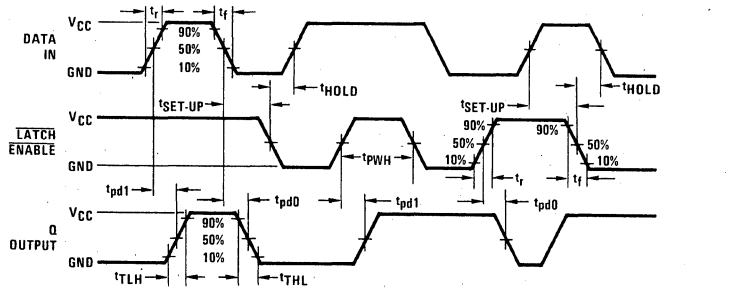






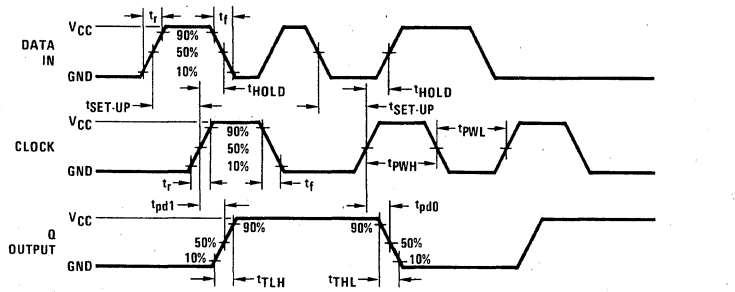
# Switching Time Waveforms

MM54C373/MM74C373



OUTPUT DISABLE = GND

MM54C374/MM74C374



OUTPUT DISABLE = GND

# DM54LS373/DM74LS373, DM54LS374/DM74LS374

## Octal D-Type Transparent Latches and Edge-Triggered Flip-Flops

### General Description

These 8-bit registers feature totem-pole TRI-STATE® outputs designed specifically for driving highly-capacitive or relatively low impedance loads. The high impedance TRI-STATE and increased high logic level drive provide these registers with the capability of being connected directly to and driving the bus lines in a bus-organized system without need for interface or pull-up components. They are particularly attractive for implementing buffer registers, I/O ports, bidirectional bus drivers, and working registers.

The 8 latches of the DM54LS373 are transparent D-type latches meaning that while the enable (G) is high the Q outputs will follow the data (D) inputs. When the enable is taken low the output will be latched at the level of the data that was set up.

The 8 flip-flops of the DM54LS374/DM74LS374 are edge-triggered D-type flip-flops. On the positive transition of the clock, the Q outputs will be set to the logic states that were set up at the D inputs.

A buffered output control input can be used to place the 8 outputs in either a normal logic state (high or low logic levels) or a high impedance state. In the high impedance state the outputs neither load nor drive the bus lines significantly.

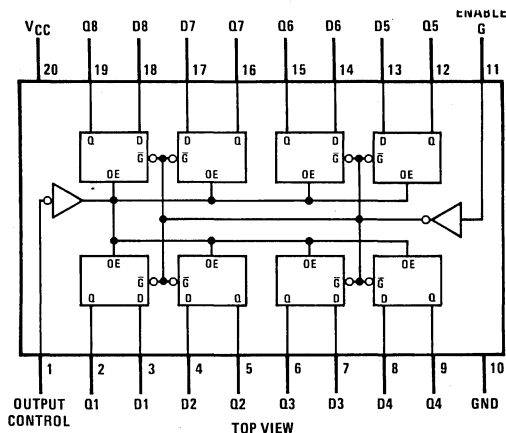
The output control does not affect the internal operation of the latches or flip-flops. That is, the old data can be retained or new data can be entered even while the outputs are OFF.

### Features

- Choice of 8 latches or 8 D-type flip-flops in a single package
- TRI-STATE bus driving outputs
- Full parallel access for loading
- Buffered control inputs
- PNP inputs reduce DC loading on data lines

### Connection Diagrams and Truth Tables

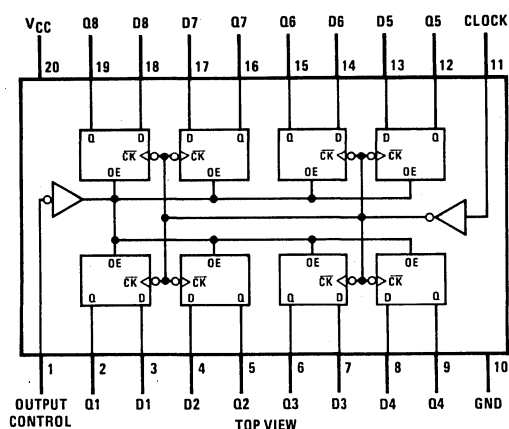
**DM54LS373/DM74LS373**  
Dual-In-Line Package



ENABLE G	D	OUTPUT
H	H	H
H	L	L
L	X	Q0

When output control is high, the output is disabled to high impedance state; however, sequential operation of these devices are not affected.

**DM54LS374/DM74LS374**  
Dual-In-Line Package



CLOCK	D	OUTPUT
↑	H	H
↑	L	L
L	X	Q0

## Absolute Maximum Ratings

Supply Voltage (Note 1)	7V
Input Voltage	7V
OFF-State Output Voltage	7V
Operating Temperature Range	
DM54LS373, DM54LS374	-55°C to +125°C
DM74LS373, DM74LS374	0°C to +70°C
Storage Temperature Range	-65°C to +150°C

## Recommended Operating Conditions

	MIN	MAX	UNITS
Supply Voltage ( $V_{CC}$ )			
DM54LS373, DM54LS374	4.5	5.5	V
DM74LS373, DM74LS374	4.75	5.25	V
High Level Output Voltage ( $V_{OH}$ )		5.5	V
High Level Output Current ( $I_{OH}$ )		-1	mA
DM54LS373, DM54LS374		-2.6	mA
DM74LS373, DM74LS374			
Width of Clock/Enable Pulse ( $t_{WY}$ )			
High	15		ns
Low	15		ns
Data Set-Up Time ( $t_{SU}$ )			
DM54LS373/DM74LS373	0↓		ns
DM54LS374/DM74LS374	20↑		ns
Data Hold Time ( $t_H$ )			
DM54LS373/DM74LS373	15↓		ns
DM54LS374/DM74LS374	5↑		ns
Temperature ( $T_A$ )			
DM54LS373, DM54LS374	-55	+125	°C
DM74LS373, DM74LS374	0	+70	°C

The arrow indicates the transition of the clock/enable input used for reference: ↑ for the low-to-high transition; ↓ for the high-to-low transition.

## Electrical Characteristics

Over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER	CONDITIONS (Note 2)	DM54LS373, DM54LS374			DM74LS373, DM74LS374			UNITS
		MIN	TYP (Note 3)	MAX	MIN	TYP (Note 3)	MAX	
$V_{IH}$ High Level Input Voltage		2			2			V
$V_{IL}$ Low Level Input Voltage				0.7			0.8	V
$V_{IK}$ Input Clamp Voltage	$V_{CC} = \text{Min}, I_I = -18 \text{ mA}$			-1.5			-1.5	V
$V_{OH}$ High Level Output Voltage	$V_{CC} = \text{Min}, V_{IH} = 2V, V_{IL} = V_{IL}(\text{MAX}), I_{OH} = \text{Max}$	2.4	3.4		2.4	3.1		V
$V_{OL}$ Low Level Output Voltage	$V_{CC} = \text{Min}, V_{IH} = 2V, I_{OL} = 12 \text{ mA}$		0.25	0.4		0.25	0.4	V
	$V_{IL} = V_{IL}(\text{MAX}), I_{OL} = 24 \text{ mA}$					0.35	0.5	V
$I_{OZH}$ OFF State Output Current, High Level Voltage Applied	$V_{CC} = \text{Max}, V_{IH} = 2V, V_O = 2.7V$			20			20	μA
$I_{OZL}$ OFF State Output Current, Low Level Voltage Applied	$V_{CC} = \text{Max}, V_{IH} = 2V, V_O = 0.4V$			-20			-20	μA
$I_I$ Input Current at Maximum Input Voltage	$V_{CC} = \text{Max}, V_I = 7V$			0.1			0.1	mA
$I_{IH}$ High Level Input Current	$V_{CC} = \text{Max}, V_I = 2.7V$			20			20	μA
$I_{IL}$ Low Level Input Current	$V_{CC} = \text{Max}, V_I = 0.4V$			-0.4			-0.4	mA
$I_{OS}$ Short Circuit Output Current (Note 4)	$V_{CC} = \text{Max}$	-30		-130	-30		-130	mA
$I_{CC}$ Supply Current	$V_{CC} = \text{Max}, \text{Output Control at } 4.5V$	DM54LS373/DM74LS373	24	40		24	40	mA
		DM54LS374/DM74LS374	27	45		27	45	mA

## Switching Characteristics $V_{CC} = 5V, T_A = 25^\circ C$

PARAMETER	FROM INPUT	TO OUTPUT	CONDITIONS	DM54LS373/ DM74LS373			DM54LS374/ DM74LS374			UNITS
				MIN	TYP	MAX	MIN	TYP	MAX	
$f_{MAX}$ Maximum Clock Frequency							35	50		MHz
$t_{PLH}$ Propagation Delay Time, Low-to-High Level Output	Data	Any Q	$C_L = 45\text{ pF}, R_L = 667\Omega,$ (Notes 5 and 6)		12	18				ns
$t_{PHL}$ Propagation Delay Time, High-to-Low Level Output	Data	Any Q			12	18				ns
$t_{PLH}$ Propagation Delay Time, Low-to-High Level Output	Clock or Enable	Any Q			20	30	16	28		ns
$t_{PHL}$ Propagation Delay Time, High-to-Low Level Output	Clock or Enable	Any Q			18	30	22	34		ns
$t_{PZH}$ Output Enable Time to High Level	Output Control	Any Q			15	28	16	28		ns
$t_{PZL}$ Output Enable Time to Low Level	Output Control	Any Q			22	36	22	28		ns
$t_{PHZ}$ Output Disable Time from High Level	Output Control	Any Q	$C_L = 5\text{ pF}, R_L = 667\Omega,$ (Note 6)		12	20	10	18		ns
$t_{PLZ}$ Output Disable Time from Low Level	Output Control	Any Q			15	25	14	24		ns

**Note 1:** Voltage values are with respect to network ground terminal.

**Note 2:** For conditions shown as min or max, use the appropriate value specified under recommended operating conditions.

**Note 3:** All typical values are at  $V_{CC} = 5V, T_A = 25^\circ C$ .

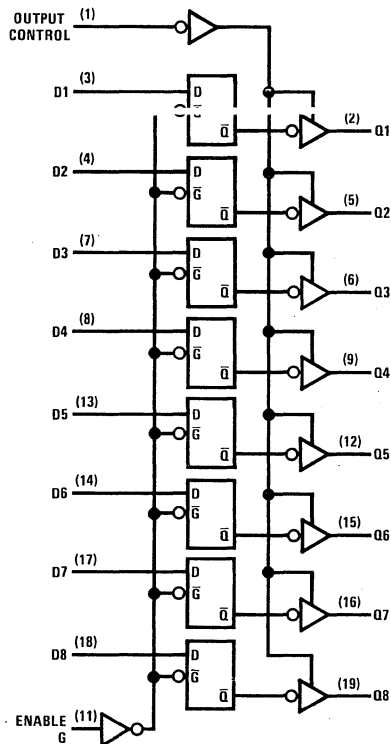
**Note 4:** Not more than one output should be shorted at a time and duration of the short circuit should not exceed one second.

**Note 5:** Maximum clock frequency is tested with all outputs loaded.

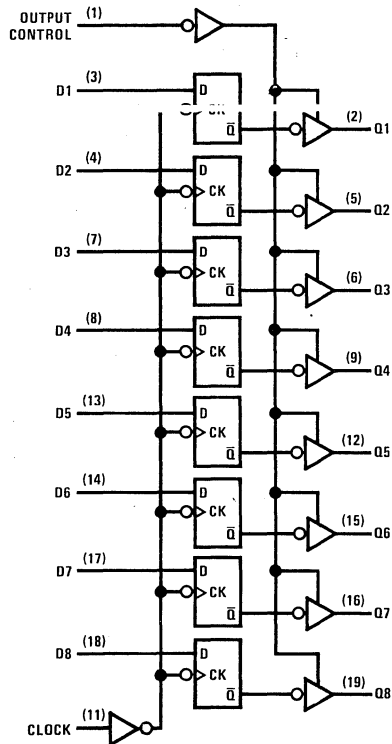
**Note 6:** See load circuits and waveforms.

## Logic Diagrams

DM54LS373/DM74LS373  
Transparent Latches



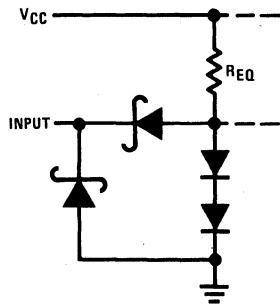
DM54LS374/DM74LS374  
Positive-Edge-Triggered Flip-Flops



# Schematic Diagrams

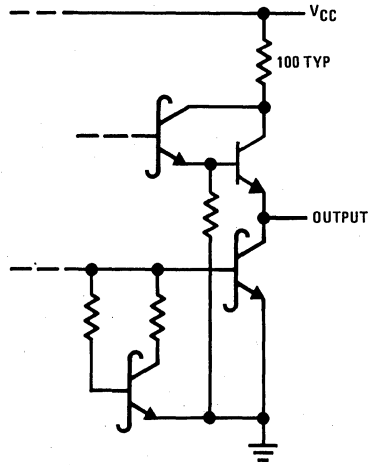
## DM54LS373/DM74LS373

Equivalent of Data, Enable, and Output Control Inputs



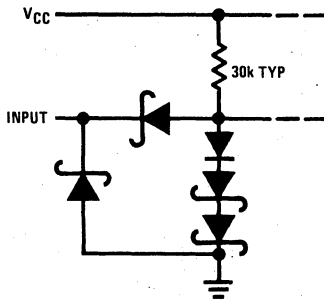
Data:  $R_{eq} = 20\text{ k}\Omega$  typ  
 Output control:  $R_{eq} = 18\text{ k}\Omega$

Typical of All Outputs

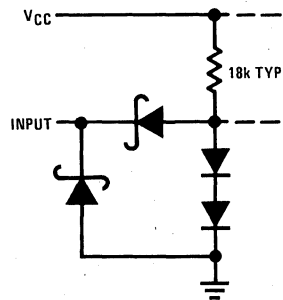


## DM54LS374/DM74LS374

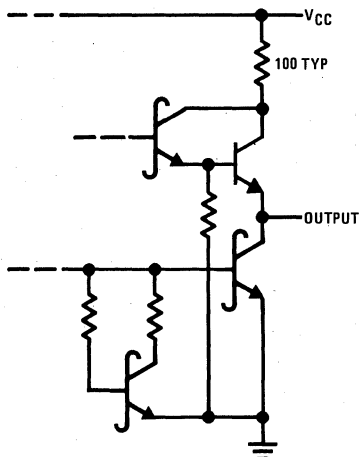
Equivalent of Data Inputs



Equivalent of Output Control Clock Inputs



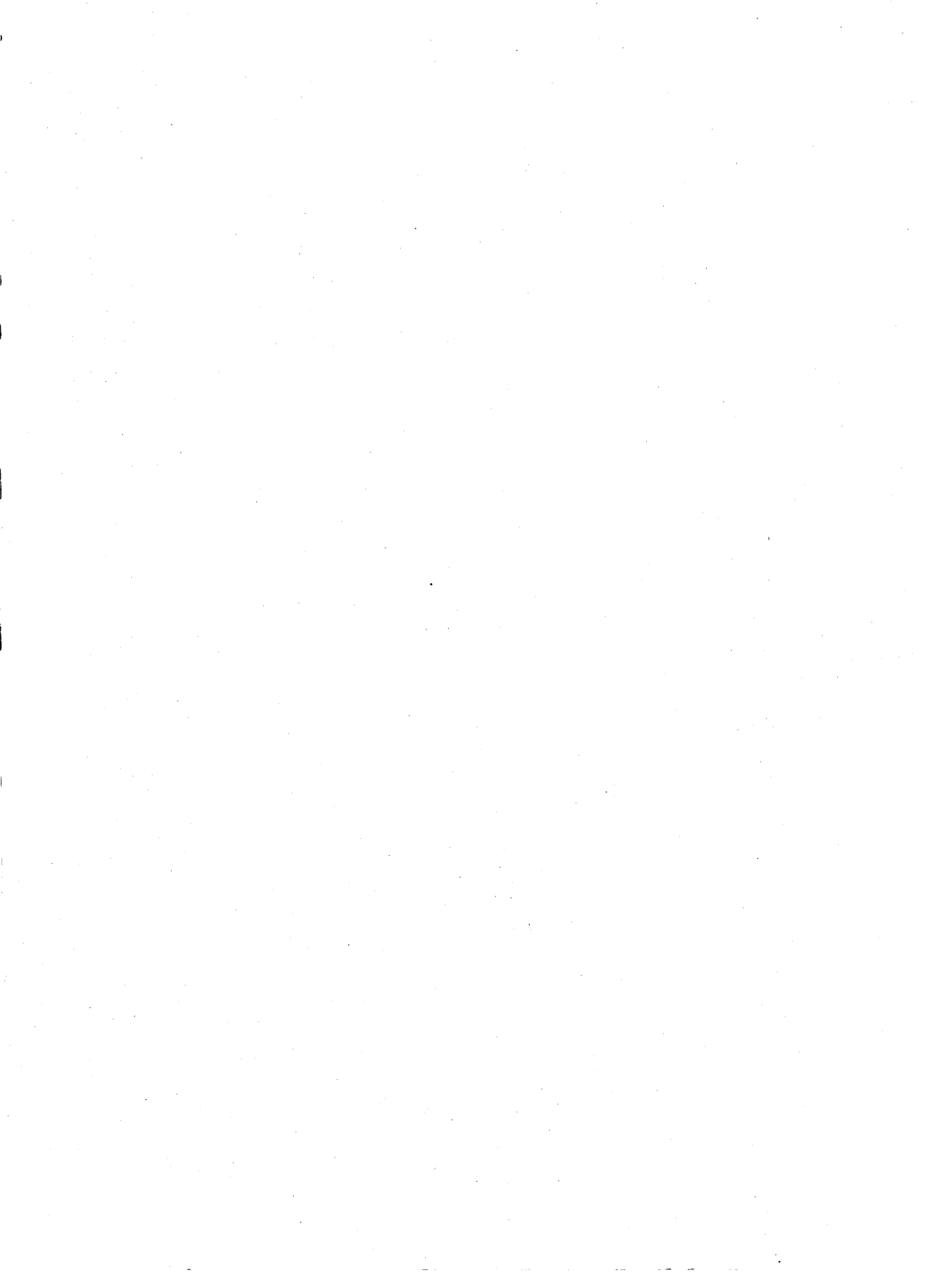
Typical of All Outputs





**Section 8**  
**Development Systems**  
**and User's Manuals**





## COP400-PDS Product Development System

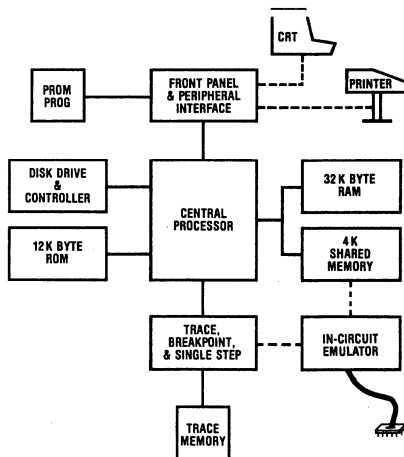
### General Description

A single development tool which supports microcontroller development activities through every phase from concept to production, the COP400 Product Development System is built around a powerful 16-bit microcomputer to allow rapid execution of sophisticated, efficient utilities. The system meets the *total* product development need. An editor and assembler are provided to handle source code entry, conversion to object code, and maintenance of documentation. The emulator card attachment allows object code to be executed under the careful control of the COPMON debug utility. A cable can be connected from the PDS to the final product; in this mode, the full power and versatility of the PDS is extended to the product-to-be for real time emulation during development. When a program is complete and ready to be committed to production, the PDS generates a transmittal disc that NSC will use to assure accurate masking of the final components. The usefulness of the PDS does not end there: a fixture is available for the incoming functional test of the ROM programmed COP400 devices. Thus, the COP400-PDS actively supports every step of a microcontroller product development activity.

### Features

- Supports the entire COP400 and COP300 microcontroller family
- A *total* concept-to-production tool
- Low cost
- 32k bytes R/W memory
- 12k bytes PROM (firmware)
- Disk-based
- Macro-assembler
- HS-232 and current loop peripheral interfaces
- Automatic baud rate selection (110-9600)
- Comprehensive in-system emulation (ISE™), with single-step, breakpoint & trace
- ROM pattern transmission by diskette

### System Diagram





# COP400 Product Development System

## FIRMWARE (ROM)

### Executive

- Entered on a power-up or initialization.
- Provides for system definition.
- Allows loading and executing disk-based programs by file name.

### Diagnostics

- Memory address, bit and word test.
- CPU operation.
- Disk read test.
- Diagnostics are called by pressing diag. button on front panel

## SOFTWARE (Disk)

### File Manager

- Create and delete disk files.
- Control file directory.
- Protect disk files.
- Pack disk files.
- Copy disk to disk.

### DSKIT

- Initialize disk.
- Disk read/write diagnostic.

### COPMON

- In-circuit emulator (ICE).
- Software breakpoints.
- External hardware breakpoint (2 lines).
- Lists user specified registers when selected breakpoint is detected.
- Real-time trace operation displays 256 address sequences. Synchronization may be pre, post, or center, and may be software or external event initiated.
- Mnemonic modification/listing of object code.
- Execution time measurement.
- Single-step.
- Step-list-restart.

### Editor

- Read text from disk or terminal.
- Display text on terminal.
- Write edited text to disk printer.
- Generate source code for assemblers.
- Extensive word processing functions.
- Edit source files larger than the edit buffer.

### Macro Assembler

- Macro generation capabilities.
- Listing controls.
- Cross reference list.
- Conditional assembly operators.
- Wide variety of directives.
- Diagnostic messages that include error position in source line.
- MASKTRN creates a disk file for transmission of customer ROM patterns and device I/O options to National Semiconductor.

## SYSTEM COMPONENTS

### COP400-PDS

- Host microprocessor.
- 32k byte R/W memory.

- Single disk drive.
- System power supply.
- 12k byte ROM (all firmware described).
- Emulator cable.
- External connectors (3).
- Two master diskettes (all software described).
- Operator's Manual.
- PROM programmer.

### Peripherals

- The COP400-PDS will interface to a wide range of peripheral devices.
- Both RS-232 and current loop ports are provided.
- Video display interface.
- Printer interface (RS-232 serial).
- Emulator cards for in-circuit emulation.

### Emulator Boards

Emulator boards are ordered separately for each of the COPS family of devices to be used. The emulator board allows real-time emulation in your applications circuitry using PROM memory or the development system R/W memory for program storage.

- PC assembly with the appropriate COP400 ROMless device and its support circuitry.
- Sockets for UV erasable PROMs.
- In-circuit emulation cable with emulated COP device plug.

## ENVIRONMENTAL REQUIREMENTS

Operating and Storage Temperature	50° to 125°F
Relative Humidity	8% to 80%
Maximum Wet Bulb Temperature	85°F

### Shipping

Temperature	40° to 125°F
Relative Humidity	8% to 80%
Maximum Wet Bulb Temperature	85°F
Weight	101 lb.

### Power

115 volts AC  $\pm$  10%, 60 Hz, fused for 600 watts  
 230 volts AC  $\pm$  10%, 50 Hz, fused for 600 watts

### Physical Specifications

Height 12½", Width 16", Depth 23"

## Ordering Information

- COP400-PDS2 — U.S. version (60 Hz, 120 VAC)
- COP400-PDS2E — European version (50 Hz, 220 VAC)
- COP400-EO2 — Emulator for 420 and 421
- COP400-EO4L — Emulator for 420L, 421L, 444L, 445L, 410L, and 411L
- COP400-E02C — Emulator for 420C, 421C, 410C, 411C
- COP400-E24 — Emulator for 440, 441, 442, 2440, 2441, 2442
- COP400-DO2 — PDS Master Diskette

## Manuals

- 420305548-002 — COP400, PDS Operator's Manual
- 420306469-001 — Emulator Cards User's Manual

# COP400 Product Development System

## User's Manual



Publication No. 420305548-002A  
Order No. 420305548-002  
October 1981

# Preface

This manual describes the COP400 Product Development System (PDS). In particular, it discusses the following topics:

- System Installation
- System Initialization
- Diagnostics
- The following PDS Programs:
  - File Manager
  - Disk Initialization and Test
  - Text File Editor
  - Cross-Assembler
  - Monitor and Debugger
  - File List
  - Cross Reference
  - Mask Transmittal
  - Memory Diagnostic
  - PROM Programmer

This manual applies only to the current COP400-PDS (Part No. COP400-PDS2). For users with the COP400-PDS with a front panel keypad, see COP400 PDS User's Manual, Publication No. 420305548-001.

This manual is for information only and is subject to change without notice.

# COP400

## Product Development System

### Table of Contents

Section	Description	Page
<b>Chapter 1. Introduction and Overview</b>		
1.1	Introduction .....	8-11
1.2	Hardware Overview .....	8-11
1.3	Software Overview .....	8-13
1.4	Emulation and Debugging Overview .....	8-14
<b>Chapter 2. Installation and Initialization</b>		
2.1	Introduction .....	8-17
2.2	Installation .....	8-17
2.3	System Initialization .....	8-17
2.4	Console Input .....	8-17
2.5	Printer Output .....	8-22
2.6	System Configuration Commands .....	8-23
2.7	Diagnostics .....	8-23
<b>Chapter 3. File Manager Program</b>		
3.1	Introduction .....	8-24
3.2	Combine Files Command .....	8-24
3.3	Copy File Command .....	8-24
3.4	Delete Command .....	8-24
3.5	Directory Command .....	8-24
3.6	Duplicate File Command .....	8-25
3.7	Duplicate Volume Command .....	8-25
3.8	Header Command .....	8-25
3.9	Locate Command .....	8-25
3.10	Pack File Command .....	8-25
3.11	Pack Volume Command .....	8-25
3.12	Protect Command .....	8-25
3.13	Rename Command .....	8-25
3.14	Space Command .....	8-25
3.15	Undelete Command .....	8-26
3.16	Volume Command .....	8-26
<b>Chapter 4. Disk Initialization and Test (DSKIT)</b>		
4.1	Introduction .....	8-27
4.2	Initialize Command .....	8-27
4.3	Address Test Command .....	8-27
4.4	Bad Sector Command .....	8-28
4.5	Clear Command .....	8-28
4.6	Directory Command .....	8-28
4.7	Dump Sector Command .....	8-28
4.8	Pattern Test Command .....	8-29
4.9	Sector Marks Command .....	8-29
4.10	Status Command .....	8-29
4.11	Test Sector Command .....	8-29

Section	Description	Page
<b>Chapter 5. Text File Editor</b>		
5.1	Introduction .....	8-30
5.2	DISK EDIT MODE .....	8-30
5.3	Invoking EDIT .....	8-30
5.4	Edit Command .....	8-30
5.5	Commands Within the Edit Window (Buffer) .....	8-30
5.6	Commands that Move the Edit Window .....	8-36
5.7	DISK EDIT MODE Setup and Quit Command .....	8-38
<b>Chapter 6. COP Cross-Assembler (ASM)</b>		
6.1	Introduction .....	8-44
6.2	The Assembly Process .....	8-45
6.3	Introduction to Assembly Language Statements .....	8-46
6.4	Assembler Statements .....	8-49
6.5	MACROS .....	8-61
6.6	Example of Creating and Assembling a User Program .....	8-67
<b>Chapter 7. COPS Monitor and Debugger</b>		
7.1	COPMON .....	8-80
7.2	Console Operation .....	8-80
7.3	COPMON Console Commands .....	8-81
<b>Chapter 8. File List Program (LIST)</b>		
8.1	Introduction .....	8-90
8.2	Invoking LIST .....	8-90
8.3	Options .....	8-90
<b>Chapter 9. Cross Reference Program</b>		
9.1	Introduction .....	8-92
9.2	Invoking XREF .....	8-92
<b>Chapter 10. Mask Transmittal Program (MASKTR)</b>		
10.1	Use of Mask Transmittal Program .....	8-93
10.2	ABORT Command .....	8-93
10.3	CHIP Command .....	8-93
10.4	COMPANY Command .....	8-93
10.5	DATE Command .....	8-93
10.6	FINISH Command .....	8-94
10.7	HELP Command .....	8-94
10.8	LIST Command .....	8-94
10.9	NAME Command .....	8-94
10.10	OPTION Command .....	8-94
10.11	PRINT Command .....	8-94
10.12	TRANSMITTAL Command .....	8-94
<b>Chapter 11. Memory Diagnostic</b>		
11.1	Use of Memory Diagnostic (MDIAG) .....	8-99

Section	Description	Page
<b>Chapter 12. COP400 PDS PROM Programmer (PROG)</b>		
12.1	Introduction .....	8-101
12.2	ALTER DATA BUFFER Command .....	8-101
12.3	BASE Command .....	8-101
12.4	CHIP Command .....	8-101
12.5	COMPARE Command .....	8-101
12.6	DEPOSIT Command .....	8-101
12.7	DUMP Command .....	8-102
12.8	ERASE Command .....	8-102
12.9	HELP Command .....	8-102
12.10	LIST Command .....	8-102
12.11	LOAD Command .....	8-102
12.12	PROGRAM Command .....	8-102
<b>Appendix</b>		
A	Sample Program .....	8-104
<b>Illustrations</b>		
Figure	Illustrations	Page
1-1	PDS as Shipped from the Factory .....	8-12
1-2	PDS Front Panel .....	8-12
1-3	PDS Rear Panel .....	8-12
1-4	PDS Rear Panel Connectors .....	8-12
1-5	PDS In-System Emulation System .....	8-14
2-1	Inserting a Diskette into the Drive .....	8-18
5-1	Operational Sequence of DISK EDIT MODE .....	8-20
5-2	DISK EDIT MODE Edit Window Operator .....	8-39
6-1	DSPLY .SRC Source Code .....	8-68
6-2	DSPLY .SRC Assembly Output Listing .....	8-73
9-1	Typical Cross Reference Listing .....	8-92

Table	Tables	Page
1-1	Recommended Peripheral Devices .....	8-13
1-2	Edge Connector Assignments .....	8-15
2-1	TTY Connector (Current Loop) .....	8-17
2-2	Printer and CRT Connectors (RS232) .....	8-17
2-3	PDS System Program Names and Prompts .....	8-19
2-4	PDS Console Input Control Characters .....	8-20
2-5	System Default Modifiers .....	8-20
2-6	PDS Internal File Types .....	8-20
2-7	Protection Levels and Safeguard Provisions .....	8-21
2-8	Disk File Error Messages .....	8-22
2-9	PDS Device Names .....	8-22
2-10	System Commands .....	8-23
2-11	Operand Parameters .....	8-23
3-1	File Manager Command Summary .....	8-26
4-1	DSKIT Command Summary .....	8-27
4-2	Command Parameter Description .....	8-27
4-3	DSKIT Command Option Description .....	8-28
5-1	Editor Commands .....	8-40
5-2	Command Format Definitions .....	8-41
5-3	Error Messages .....	8-42
5-4	Edit Command Control Characters .....	8-43
6-1	ASM Arithmetic and Logical Operators .....	8-48
6-2	COP400 Instruction Set .....	8-51
6-3	COP400 Instruction Set Table Symbols .....	8-54
6-4	Alphabetical Mnemonic Index of COP400 Instructions .....	8-54
6-5	Table of COP400 Instructions Listed by OP Codes (Hexadecimal) .....	8-55
6-6	Summary of Assembler Directives .....	8-57
6-7	List Options .....	8-57
6-8	ASCII Character Set in Hexadecimal Representation .....	8-58
6-9	Display Digit Segments .....	8-59
7-1	Valid Chip Numbers .....	8-81
7-2	Summary of COPMON Console Commands .....	8-87
7-3	Operand Syntax .....	8-88
7-4	GO Operation Summary .....	8-89
8-1	Summary of List Options .....	8-90
10-1	MASKTR Command Summary .....	8-97
12-1	PROM Programmer Command Summary .....	8-101
12-2	Summary of PROG Operands .....	8-103

# Introduction and Overview

## 1.1 General Description

The COP400 Product Development System (PDS) is designed to aid in the development of products using National Semiconductor's COP400 Microprocessor series.

The PDS is a disk-oriented system. It is capable of creating and accessing data and program files stored on a floppy diskette. This allows fast and easy access to system software, rapid access to program files, and a convenient way to provide National Semiconductor with program data for the mask-making process.

The PDS provides for debugging of the COP400 device. Debugging uses hardware and software to single-step through a COP400 program, breakpoint on an address, trace program execution, and dump out internal COP400 registers. This feature speeds up the development cycle.

The user interacts with the PDS via a system console such as a teletype or CMI. An optional printer can be attached to obtain program listings quickly. The PDS front panel allows the user to perform specific development functions without a system console. Connectors on the rear panel of PDS can be connected to an emulator board, which emulates a COP400 chip in the user's system. A PROM programmer on the PDS front panel allows the emulator board to be portable, for emulation in the final environment of the user's system.

The following sections provide an overview of the COP400 PDS Hardware, COP400 PDS Software, and Emulation and Debugging.

## 1.2 Hardware Overview

This section provides a general description of the PDS hardware consisting of the following:

- Front and Rear Panels
- Peripheral Devices

Figure 1-1 shows the PDS as shipped from the factory.

### 1.2.1 Front and Rear Panels

The PDS front panel is shown in Figure 1-2. The switch in the lower right corner is the power switch. To the left of the power switch are five switches. Two switches, labeled PROGRAM LOAD, are used to load and execute the COP Monitor discussed in Chapter 7. A third switch, labeled DIAG, is for a diagnostic test on the PDS internal memory and the disk drive. A fourth switch, labeled INIT, is for PDS system initialization. A fifth switch, labeled AUX, is a spare, not currently used by the PDS. The five switches are discussed in more detail in Chapter 2.

In the center of the PDS front panel is a quick-release socket that is used for programming MM2758, MM2716, MM2732, MM2724 EPROMs.

On the left side of the PDS front panel is the floppy disk drive door. The door latches are closed and opened with the rectangular button to the left of it. In the center of the button is an indicator light which lights when the drive is in use.

The PDS rear panel is shown in Figure 1-3. On each side there is a cooling fan filter screen. Below the left screen is a fuse holder and a plug for the power cable. Above this screen are six connectors used to connect peripheral devices to PDS.

### 1.2.2 Peripheral Devices

Peripheral devices provide user interface with the PDS. A console is required for entering commands. A console may be a CRT or TTY, i.e., any device with an RS232 or a current loop interface. A printer for producing hard copy output is required when using a CRT, and optional when using a TTY. A TTY provides its own hard copy output. An emulator board is required for user's system emulation. Emulator Boards are discussed in the In-System Emulator Boards Manual, Publication No. 420306469.

There are six connectors on the PDS rear panel used to connect peripheral devices to the PDS, see Figure 1-4 for a close-up view. TTY is a 9-pin connector for teletype or other current loop device connection. CRT is a standard 25-pin RS232 connector for a CRT or other RS232 device. PRINTER is a standard 25-pin RS232 connector used to connect a printer to the PDS.

The console device (CRT or TTY) may operate at one of the following baud rates: 110, 150, 300, 600, 1200, 2400, 4800 or 9600. The user can set EVEN or NO parity, and carriage return and line feed delays from 0 to 1000ms for the console. The recommended console set up for the various allowable baud rates are as follows:

110 Baud:

- 8-bit data (No Parity — PDS resets bit 8 = 0),
- 2 Stop bits, Full Duplex operation

150-9600 Baud:

- 8-bit data (No Parity — PDS resets bit 8 = 0),
- 1 Stop bit, Full Duplex operation; or 7-bit data,
- Even Parity, 1 Stop bit, Full Duplex operation.

If the console uses a current loop interface, PDS will assume a 110 Baud rate with the set-up as shown above.

The printer device must meet the same requirements listed above for a console. The user can set EVEN or NO parity, and carriage return, line feed, form feed, and vertical tab delays from 0 to 1000ms for the printer. Table 1-1 lists some typical peripheral devices.



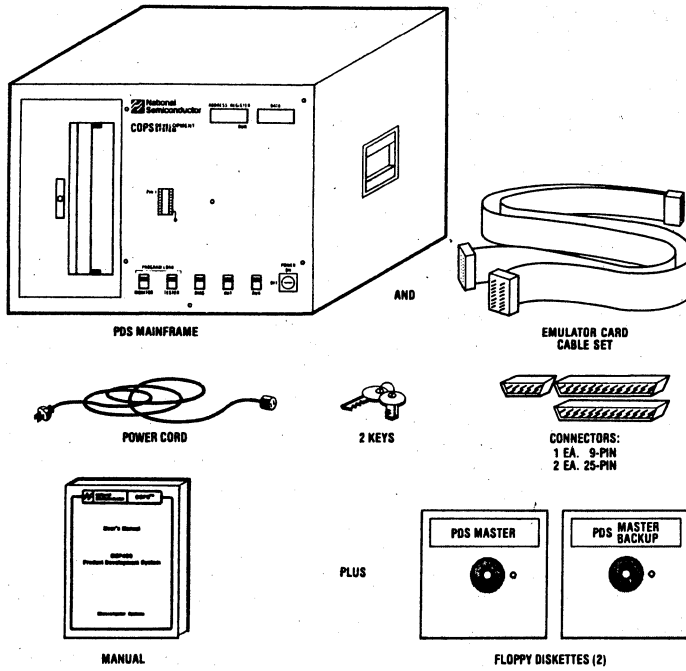


Figure 1-1. PDS as Shipped from the Factory

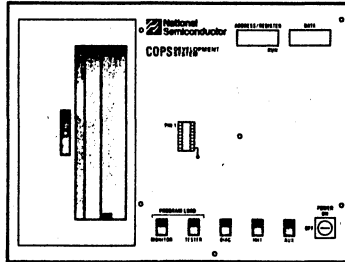


Figure 1-2. PDS Front Panel

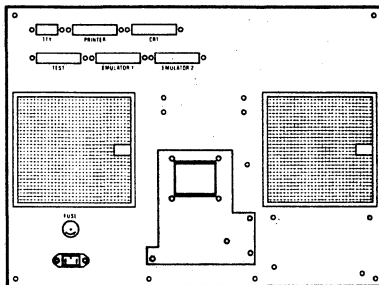


Figure 1-3. PDS Rear Panel

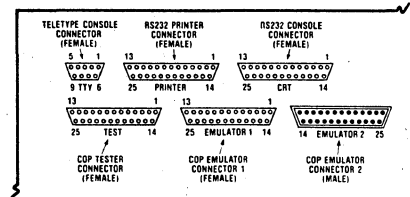


Figure 1-4. PDS Rear Panel Connectors

### 1.3 COP400 PDS Software Overview

PDS software is divided into two parts:

- Firmware in ROM
- Software on disk

The firmware contains general routines for console and printer configuration, system initialization, diagnostics, and program loading. The user invokes a general routine by entering a command name at the console or by pressing one of the five switches on the PDS front panel.

The software contains the programs for file editing, assembly, debugging, and PROM programming. The PDS Programs are interactive programs. The user invokes the program by entering the program name at the console, followed by a carriage return. The system responds with a message and prompts for user-entered commands. Each program has several commands.

This section gives an overview of the following COP400 commands and programs:

- System Configuration and Diagnostic Commands
- File Manager Program (FM)
- Disk Initialization and Test Program (DSKIT)
- Text File Editor (EDIT)
- COP Cross-Assembler (ASM)
- COPS™ Monitor and Debugger (COPMON)
- File List Program (LIST)
- Cross Reference Program (XREF)
- Mask Transmittal Program (MASKTR)
- Memory Diagnostic (MDIAG)
- PROM Programmer (PROG)

#### 1.3.1 System Configuration and Diagnostic Commands

The system configuration commands are used to configure the console. The diagnostic command performs a short diagnostic routine on the internal memory and the disk drive. The System Configuration commands and the Diagnostics are described in Chapter 2.

Table 1-1. Recommended Peripheral Devices

Device	Vendors
CRT:	<ol style="list-style-type: none"> <li>1. Lear Siegler Model ADM-3, Part No. 129450 Lear Siegler 714 No. Brookhurst St. Anaheim, CA 92803</li> <li>2. Hazeltine Model 1500 Hazeltine Industrial Products Div. Greenlawn, NY 11740</li> </ol>
PRINTER:	<ol style="list-style-type: none"> <li>1. Centronics Mod. 702 w/RS232 interface Centronics Data Computer Corp. Hudson, NH 03051</li> <li>2. FACIT Mod. 4555 w/RS232 interface (Sweden)</li> <li>3. G.E. TERMINET w/RS232 interface</li> </ol>
TTY:	<ol style="list-style-type: none"> <li>1. Teletype Mod. ASR3320/3JC manual read</li> <li>2. Decwriter</li> <li>3. Silent 700</li> </ol>

Note: A Silent 700 with RS232 interface requires pins 5 and 8 to be connected together.

#### 1.3.2 File Manager Program (FM)

The File Manager (FM) is a PDS system program that provides an interface to disk files. FM enables the user to copy files, delete and undelete files, list the disk directory, duplicate disks, list file size and type, list space available on a disk, list and change the disk name and perform various other functions. The File Manager Program is described in Chapter 3.

#### 1.3.3 Disk Initialization and Test Program (DSKIT)

The Disk Initialization and Test Program is a PDS system program that initializes new disks. Initialization consists of writing sector sync marks on each sector, writing and verifying a test pattern on each sector, writing the volume name and header onto the disk, and create an empty directory. The Disk Initialization and Test Program is described in Chapter 4.

#### 1.3.4 Text File Editor (EDIT)

The Text File Editor (EDIT) is a system program which creates or changes text files. The Text File Editor can insert, delete, alter, and list program text as well as write the text to a floppy disk. EDIT can accept source from either disk files or console entry. The Text File Editor is described in Chapter 5.

#### 1.3.5 COPS Cross-Assembler (ASM)

The COPS Cross-Assembler (ASM) is a PDS system program which translates symbolic program files (created with the Text File Editor) into object code files containing program instruction in machine language. The COPS Cross-Assembler also generates output listings containing source statements, corresponding machine code and memory locations, and error messages. The COPS Cross-Assembler is described in Chapter 6.

#### 1.3.6 COPS Monitor and Debugger (COPMON)

The COPS Monitor and Debugger (COPMON) is a PDS system program which can monitor the execution of programs. The COPMON program permits program tracing and examination and modification of system registers during program execution. The COPS Monitor and Debugger is described in Chapter 7.

#### 1.3.7 File List Program (LIST)

The File List Program (LIST) is a PDS system program which lists files on the system console or printer. LIST has several printing options. The File List Program is described in Chapter 8.

#### 1.3.8 Cross Reference Program (XREF)

The Cross Reference Program (XREF) is a PDS system program which prints a symbol map of COP assembly language programs. The symbol map shows the name of every symbol in the program, the line number where the symbol is defined, and all of the line numbers where the symbol is used. The Cross Reference Program is described in Chapter 9.

### 1.3.9 Mask Transmittal Program (MASKTR)

The Mask Transmittal Program (MASKTR) is a PDS system program which creates the Transmittal File used by National Semiconductor to create the COP chip ROM/OPTIONS mask. The Mask Transmittal Program is described in Chapter 10.

### 1.3.10 Memory Diagnostic (MDIAG)

The Memory Diagnostic (MDIAG) is a PDS system program which runs diagnostics on PDS memory. This program will run ADDRESS, BIT, WORD, and GALPAT tests. The Memory Diagnostics Program is described in Chapter 11.

### 1.3.11 COP400 PDS PROM Programmer (PROG)

The COP400 PDS PROM Programmer (PROG) is a PDS system program which operates the PROM programmer located in the center of the PDS front panel. The PROM programmer programs MM2716, MM2732, MM2724, and MM2758 EPROMs. The COP400 PDS PROM Programmer is described in Chapter 12.

## 1.4 COP400 Emulation and Debugging Overview

The following COP400 Emulation and Debugging facilitate COP400 system development:

- In-Circuit Emulation
- Trace
- Breakpoint
- Single-Step

Emulation and Debugging commands are described in Chapter 7.

### 1.4.1 In-Circuit Emulation

In-Circuit Emulation refers to execution and testing of COP400 programs while under PDS control. The PDS In-Circuit Emulator emulates the operation of the user's COP400 system and permits user programs to be tested in the user environment. The user may modify and re-test programs if errors are found. This ensures that the program is correct before dedicating it to mask-making.

The PDS In-Circuit Emulation System is shown in Figure 1-5. The Emulator Card emulates the COP chip by using a special 400 device which is identical to a masked-ROM COP. The ROM of the special COP400 device has been replaced by a connection to an external memory. The external memory may consist of PROMs which plug into the emulator card, or random access memory located within the PDS. Random access memory used by both the emulator card and PDS is called "shared memory."

The contents of shared memory may be first loaded from a disk file, then altered and/or listed using the PDS system software (COPMON) described in Chapter 7.

A TARGET cable, supplied with the PDS, connects the PDS to the Emulator card. One end of the TARGET cable attaches to the 50-pin edge connector on the card. The other end splits into two connectors, one male and one female, that attach to the PDS rear panel connectors labeled "EMULATOR 1 and 2." Table 1-2 shows the wiring of these connectors. Five types of signals come across the TARGET cable:

1. Shared memory address and data lines — used by the emulator card to access shared memory.
2. +5 V<sub>DC</sub> and GND power supply lines — used to power the emulator card. These lines are from the PDS power supply and should never be used to power the user's system.
3. RESET line — permits PDS software reset of the emulator card (see Chapter 7).
4. External event lines — permit breakpoint and single-step on signals from external devices. Signals must be TTL compatible.
5. Trigger out line — permits the PDS to signal external devices such as oscilloscopes. The signal is TTL compatible.

The Emulator Card is described in detail in the In-Circuit Emulator Cards Manual, Publication No. 420306469.

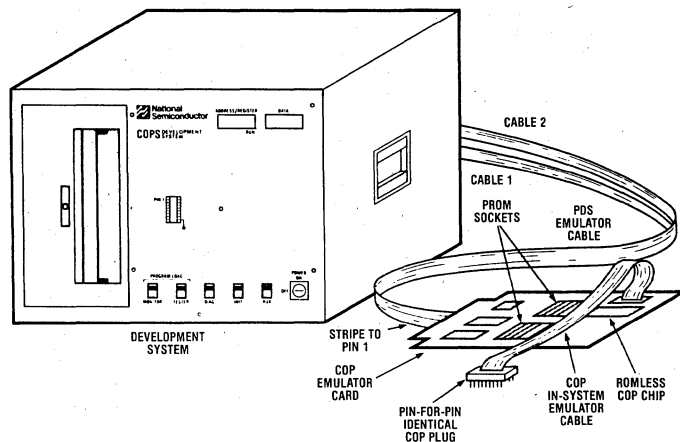


Figure 1-5. PDS In-System Emulation System

Table 1-2. Edge Connector Assignments

Connector No.	Name	Description
1	GND	Signal and power return
2	GND	Signal and power return
3	V <sub>CC</sub>	+5V <sub>DC</sub> power from Development System
4	V <sub>CC</sub>	+5V <sub>DC</sub> power from Development System
5	EX2	Buffered External Event
6	EX1	Buffered External Event
7	EX4	Buffered External Event
8	EX3	Buffered External Event
9	CLK	Buffered AD/DATA signal from COP4XX
10	SKIP	COP4XX skip status line
11	A8	COP4XX program counter address bit
12	A9	Address Bit
13	A3	Address Bit
14	A7	Address Bit
15	A1	Address Bit
16	A2	Address Bit
17	A4	Address Bit
18	A0	Least significant address bit
19	A6	Address Bit
20	A5	Address Bit
21	Not Used	
22	A10	Most significant address bit
23	Not Used	
24	Not Used	
25	Not Used	
26	Not Used	
27	Not Used	
28	Not Used	
29	Not Used	
30	Not Used	
31	Not Used	
32	Not Used	
33	B0	Least significant COP object code bit
34	B7	Most significant COP object code bit
35	B2	Object code bit
36	B5	Object code bit
37	B3	Object code bit
38	B4	Object code bit
39	B6	Object code bit
40	B1	Object code bit
41	TRIGGER OUT	BREAKPOINT/TRACE indicator
42	Not Used	
43	RST	Same as RESET
44	PROM DISABLE	Select PROM or Shared Memory mode
45	See Note 1	
46	See Note 1	
47	V <sub>CC</sub>	+5V <sub>DC</sub> power from Development System
48	V <sub>CC</sub>	+5V <sub>DC</sub> power from Development System
49	GND	Power and signal return
50	GND	Power and signal return

Note 1: Pins 45 and 46 are used as follows:

PDS with target board 980306552 REV A or later, normally not used.

PDS with target board 980305551 REV F or earlier, -12V<sub>DC</sub> from the PDS.

### 1.4.2 Trace

A trace records the path of execution control through the user program.

A trace may store up to 254 consecutive COP instruction addresses in the PDS Trace memory. A trace can be initiated on user command or it can be set up by the user to initiate when one of these conditions occurs:

1. The COP chip program counter attains a specific address.
2. External Events 1 and 2 attain specific values.

The user can specify that a given number of occurrences (from 1 to 256) of the above conditions must occur before trace is initiated. At the time that trace is initiated, a positive edge occurs on the Trigger Out (T.O.) signal post on the emulator card. This signal can be used for triggering oscilloscopes or logic analyzers.

The user may specify the number of COP instruction addresses that are to be stored prior to the trigger. This number may be from 0 to 253. The remainder of trace memory will automatically store as many instruction addresses as possible following the trigger. The user can thus perform pre-triggering, post-triggering, and mid-triggering.

In addition to COP instruction addresses, trace memory stores the following data:

1. COP chip SKIP flag, which indicates whether or not the corresponding instruction was skipped.
2. The four external event signals connected by the user to the emulator card posts labeled 1,2,3, and 4.

The user can specify that a given number of occurrences (from 1 to 256) of the above conditions must

occur before a break is initiated. At the time that break is initiated, a positive edge occurs on the Trigger Out (T.O.) signal post on the emulator card. This signal can be used for triggering oscilloscopes or logic analyzers.

When break is initiated, the COP chip instruction lines are switched from shared memory or PROMs over to a special transparent memory containing a dump program. This program causes the special COP400 chip on the emulator card to dump all internal registers and memory to PDS, where it is available for inspection by the user. The program then restores all registers and maintains the COP chip in a waiting state until commanded by the user to continue normal program execution.

### 1.4.3 Breakpoint

Breakpoint provides a means to examine internal COP registers at specific points within program execution. A break can be initiated immediately on user command, or it can be set up by the user to initiate automatically when one of the following conditions occurs:

1. The COP chip program counter attains a specific address.
2. External Events 1 and 2 attain specific values.

### 1.4.4 Single-Step

Single-step provides a means for single-stepping the COP chip by breakpointing on each consecutive instruction. Internal COP registers are available to the user after each step.

# PDS Installation and Initialization

## 2.1 Introduction

This chapter provides a description of PDS installation and initialization procedures. Also discussed is the console input, including command syntax, printer output, system configuration, diagnostics and error messages.

## 2.2 PDS Installation

Installation of the PDS involves a physical check of the PDS, connection of the peripheral devices to the PDS, and application of power to both the PDS and the peripheral devices. To install the PDS, do the following:

1. Remove the PDS top cover by removing the two screws located toward the rear of the cover and sliding the cover off. Make sure that all six PC boards are seated and firmly fastened. Replace the top cover.
2. Connect the peripheral devices to the PDS. The user must provide appropriate cables for connection between the peripheral devices and the connectors on the PDS rear panel. Pin assignments are shown in Tables 2-1 and 2-2.
3. Plug the power cable into the rear of the PDS. The PDS is now ready for operation.

## 2.3 System Initialization

System Initialization involves powering up the PDS and configuring system peripherals. To initialize the system, do the following:

1. Turn on power to the PDS and system peripheral devices. (The PDS is powered up using the front panel key switch.) The system displays

CR?

on the front LED display.

2. Press the carriage return key on the system console. The system performs an initialization routine, sets the console baud rate and type (RS232 or current loop), and then displays message:

EXEC Rev A

E>

at the console. The system is now in the executive (EXEC) program. E> is the program prompt.

3. Configure console and printer operation using the System Configuration Commands (see Section 2.6). Note that console configuration can be skipped if the default configuration is sufficient. Default configuration for current loop interface is 110 baud, no parity, two stop bits, and full duplex operation. Default for RS232 is 1200 baud, no parity, and no line feed or carriage return delays.
4. Insert the PDS MASTER diskette in the PDS floppy disk drive and close the door (see Figure 2-1).

The PDS is now ready to accept PDS commands. The user may enter a command or program name at the console as described in the next section or press the PDS front panel switches MONITOR, DIAG, and INIT.

Table 2-1. TTY Connector (Current Loop)

Pin Number	Signal Name
1	TTY Xmitter (+)
2	TTY Printer (+)
3	Reader Relay (+)
4	Not connected
5	Not connected
6	TTY Xmitter Return (-)
7	TTY Printer Return (-)
8	Reader Relay Return (-)
9	Not connected

Table 2-2. Printer and CRT Connectors (RS232)

RS232 Data Set Pin No.	Signal Name	Printer Pin No.	CRT Pin No.
1	Chassis Ground	1	1
2	Transmitted Data	2 not conn.	2
3	Received Data	3	3
4	Request to Send	4	4 not conn.
5	Clear to Send	5	5
6	Data Set Ready	6	6
7	Signal Ground	7	7
8-19		8-19 not conn.	8-19 not conn.
20	Data Terminal Ready	20	20 not conn.
21-25		21-25 not conn.	21-25 not conn.

The MONITOR switch loads the COPMON program described in Chapter 7.

The DIAG switch loads the Diagnostic routine described in Section 2.7.

The INIT switch resets the PDS. The user may then initialize the PDS as described above.

## 2.4 Console Input

### 2.4.1 Commands and Command Line Syntax

The PDS commands cause the system to perform a specified operation or load a PDS program from diskette into memory. A PDS command consists of symbols, names, and operands that specify the command type and the operation to be performed. The operands specify the diskette files, numbers, names, and options that are to be used during command operation.

The user enters PDS commands at the command line. The command line is a line at the system console containing a program prompt. A prompt consists of a letter followed by the ">" symbol, e.g., "E>" is the EXEC program prompt. The prompt indicates which program the system is currently executing. Each PDS program has a unique prompt. Table 2-3 lists the prompts of the various PDS programs.

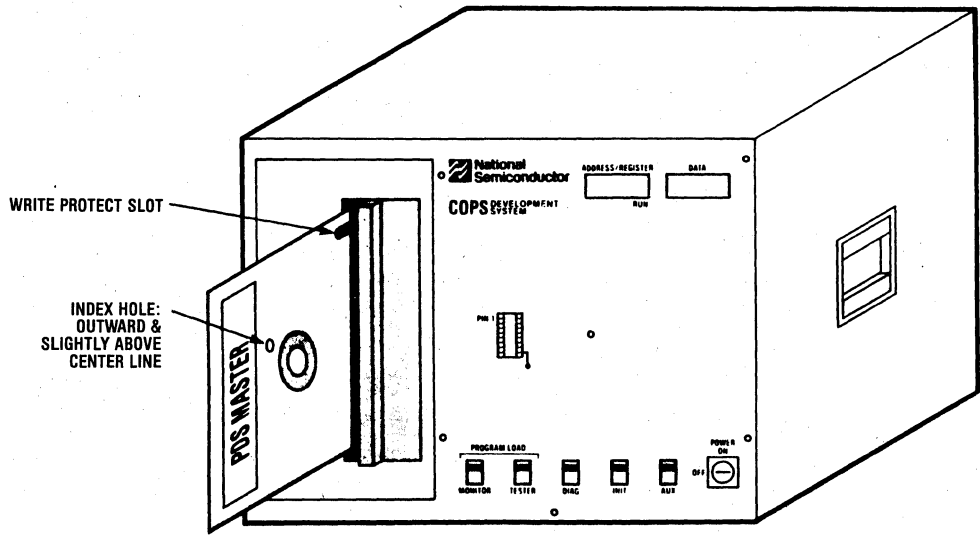


Figure 2-1. Inserting a Diskette into the Drive

Table 2-3. PDS System Program Names and Prompts

System Program Name	Function	Prompt
ASM*	COP Macro Assembler	A>
COPMON*	COP Monitor	C>
DSKIT*	Disk Initialization and Test	D>
EDIT*	Text File Editor	E>
FM*	File Manager Program	F>
LIST*	Text File Listing	L>
MDIAG*	PDS Memory Diagnostic Program	M>
PROG*	PROM Programming Utility	P>
XREF*	COP Program Cross Reference	R>
MASKTR*	Mask Transmittal Program	T>
EXEC**	PDS Executive Program	X>

\* System program on Master Diskette.

\*\* System program in Firmware.

When a prompt appears, the system is ready to accept a command. The PDS commands are divided into three types:

- System Configuration Commands
- Program Invocation Commands
- Program Commands

The System Configuration commands configure the system peripheral devices. The commands consist of two "at signs" (@ @), a command name, and command operands. Commands must be terminated by a carriage return.

The Program Invocation commands load PDS programs from diskette in memory and change the current program prompt. The commands consist of an "at sign" (@), a program name, and program operands and must be terminated by a carriage return.

The Program commands cause the current PDS program to perform a specific operation. The commands consist of a command name and command operands and must be terminated by a carriage return. Each PDS program has a unique set of Program commands.

The syntax of a PDS command depends on the command function and/or the program to which it belongs. In describing command syntax, the following conventions are used. Upper-case and lower-case letters are used in these conventions; any combination of upper-case and lower-case letters may be used when actually entering the commands.

UPPER-CASE letters show the command names and keywords. Mandatory items are shown outside of the brackets [ ]; they must be included in the command.

If an item shown consists of underscored letters followed by non-underscored letters, then that item may be entered in an abbreviated form. Minimum legal abbreviation of such items is the underscored letters portion; in addition, any number of the non-underscored letters that follow may also be used.

Blanks or commas, when present in command strings, are significant; they must be entered as shown. Multiple blanks may be used in place of a single blank.

< >—angle brackets enclose descriptive names (in lower-case) for user-supplied names/labels for commands, parameters, devices, and files.

{ }—braces enclose more than one item out of which one, and only one, must be used. The items are separated from each other by a logical OR sign "|".

[ ]—brackets enclose optional items(s). Brackets within a bracket enclose item(s) which may be optionally entered only if the item outside that inner bracket is entered.

|—logical OR sign separates items out of which one and only one may be used.

...—three consecutive periods indicate optional repetition of the preceding item. If a comma precedes the three periods, then each item must be separated from the other by a comma.

#### 2.4.2 Control Characters

PDS uses a console input routine which has several features that allow the user to correct typing mistakes. Among the features are the ability to backspace and to abort a line using control characters. Table 2-4 describes the various control characters and the function of each one. These control characters can be used at any time when the user is typing on the PDS console. If a hardcopy console is being used, most of the control characters cannot be used because of the inability to back up and change characters that have already been typed. However, the Shift/O, Control/Q, Control/I, and carriage return characters can be used.

#### 2.4.3 Disk Files

A disk file is a collection of data stored on a disk and given a name. (The words disk, diskette, and disc are used interchangeably throughout this manual.) A PDS filename has the following syntax:

```
[ <volume name> : ] <name> [ . <modifier> ]
```

The brackets ([ ]) around a term indicate that the term is optional and may be left off. An example of a filename is PDS:SAMPLE.SRC. The volume name is PDS, the name is SAMPLE, and the modifier is SRC.

The volume name is a name given to a diskette. All files on a particular diskette have the same volume name. The volume name is given to the diskette when it is initialized (see Chapter 4) and can be changed with the File Manager program (see Chapter 3). It consists of one to eight alphanumeric characters—blanks and special characters are not permitted. The volume name is optional in a filename. If PDS encounters a filename with no volume name, it will use the volume name of the diskette that is currently in the PDS disk drive. If given, the volume name must be separated from the remainder of the filename by a colon.



Table 2-4. PDS Console Input Control Characters

Character	Function
Control/H	Backspace one character, but do not delete the character that is backspaced over.
Shift/0 ("←" on some TTY)	Delete one character back.
Control/Q	Abort line and try again.
Carriage return	Line is completed. Must be entered at end of each line.
Control/T or Control/I	Tab. (See @ @TAB Command for setting tabs.)
Control/X	Delete character at current CRT cursor position.
Control/L	Forward space one character.
Control/A	Insert characters before current cursor character.
Control/B	Backspace one word.
Control/F	Forward space one word.
Control/C	Forward space to third tab position (for comments).
Control/D	Same as carriage return except line is truncated at current cursor position.
Control/E	Forward space to end of line.
Control/S	Backspace to start of line.
Control/O or Control/Z	Forward space to next occurrence of next character typed.
Control/P or Control/W	Forward space one character beyond next occurrence of next character typed.

Note: If no characters have been typed on a line, forward spacing will space over the last line typed, a useful means of repeating the last line. If the last line ended in "PR", it will not appear on the repeated line.

The name part of a filename may consist of one to eight alphanumeric characters. The first character must be an alphabetic—blanks and special characters are not permitted.

The modifier part of a filename may consist of up to three alphanumeric characters—blanks and special characters are not permitted. It is separated from the beginning part of the filename by a period. A period with no character following it specifies a modifier with zero characters. The modifier is usually used to describe the type of a file. For example, SRC is used for text files and MP is used for PDS system program files. This convention is not mandatory. The user may choose any modifier. The modifier and its preceding period are optional. If left off, PDS will provide a default modifier. Table 2-5 lists the default modifiers.

Table 2-5. System Default Modifiers

Modifier	Definition
SRC	Source Text
LM	COP Load Module
MP	PDS System Program
LST	Listing File
SYT	Special System File
TRN	PDS Transmittal File

Each file on a diskette has a unique NAME.MODIFIER combination. The user creates files using the PDS file manager, text editor, or assembler programs. PDS maintains a directory on each diskette, describing the name and other information for each file on it. The directory can be listed by using the PDS File Manager program (see Chapter 3).

Each file has a special number called an Internal File Type (IFT) maintained by PDS in the diskette directory. The IFT is not alterable by the user. It is used by PDS to indicate the type of data in each file (source text, system program data, etc.). This allows PDS to prevent the user from accidentally assembling a binary data file, or attempting to execute a source text file. The IFT is not related to the file modifier. The modifier is selected by the user; the PDS selects the correct IFT regardless of what modifier is used. Table 2-6 lists the PDS IFTs.

Table 2-6. PDS Internal File Types

File Type	Definition
SYM	Symbolic Text
LM	COP Load Module
MP	PDS System Program

A file whose IFT is SYM (symbolic) consists of ASCII data written on the disk. The PDS File Manager program generates the SYM file type when copying ASCII data to the disk or when copying another SYM file. The PDS Text Editor program (see Chapter 5) requires a SYM file when reading data from the disk, and generates a SYM file when writing data to the disk. The PDS Assembler program requires a SYM file as input, and generates a SYM file when creating a listing file.

A file whose IFT is LM (Load Module) consists of binary data in COP load module format. The PDS Assembler program (ASM) generates an LM file as object code output. The COP Monitor program (COPMON) requires an LM file for loading into shared memory.

A file whose IFT is MP (Main Program) consists of binary data in a format that allows it to be executed by PDS with an @ command, described later in this chapter. The PDS programs FM and Edit are examples of this file type.

PDS maintains another number for each file, called a protection level. This is used to prevent accidental destruction of files. Table 2-7 is a list of protection levels and their safeguard provisions. System programs such as FM and Edit create files for the user as level 2 files. All system programs are initially level 3 files. The protection level of any file can be changed with the File Manager PROTECT command (see Section 3.12).

If PDS is directed to write into an existing file, it will delete the existing file and recreate a new file of the same name, type, and protection level. A file cannot be recreated if its protection level is 3, and if its protection level is 2, the user must give permission for recreation. A deleted file is not removed from the diskette. It still exists and can be undeleted with the File Manager UNDELETE command, provided that the disk has not been packed (see Section 3.15).

A third number for each file, called the version number, is set to 1 the first time the file is created. Each time the file is recreated, as, for example, when a text file is edited using the editor program, the version number is incremented. This number is to keep an up-to-date backup of a file. It is recommended that the user always keep a backup of every file, because diskettes go bad occasionally. The File Manager DUPLICATE command is used to back up a file (see Section 3.6).

A diskette is divided into sectors. There are 616 sectors on each diskette. One sector will hold approximately

20 average lines of text. The diskette directory requires at least eight sectors of its own. The File Manager DIRECTORY command can be used to list the size of each file (see Section 3.5).

**Table 2-7.**  
**Protection Levels and Safeguard Provisions**

Protection Level	User Notified of Creation?	User Approval Required to Delete or Modify File?
0	No	No
1	Yes	No
2	Yes	Yes
3	Yes	Delete/Modify not allowed

The PDS disk file manipulation routines will generate an error message when certain conditions occur. A file error message has the following format:

```
DISK ERROR, FILE <filename>
error message 1
[error message 2]
```

Table 2-8 is a list of the error messages and their meanings. Normally only the first nine messages given in the table will occur. In some messages there is no filename involved, in which case only ":" will be printed for the filename. Sometimes two error messages will be printed.

Table 2-8. Disk File Error Messages

Error Message	Problem
WRONG DISK VOLUME	User referred to a file on a diskette other than the one in the drive.
DRV NOT RDY	No disk in drive, drive door isn't shut, or diskette is jammed.
FILENAME SYNTAX	User typed an illegal filename.
END OF FILE	User tried to read past the end of the file while using the text editor.
END OF DISK	Diskette is full, no more data can be stored on it. See WARNING in Chapter 7 concerning this error.
CANT DELETE	Attempt to delete a file whose protect level is 3, or user didn't give permission to delete a file whose protect level is 2.
ILLEGAL DEVICE	User referred to an illegal device.
FILE NOT FOUND	Reference was made to a file that is not on the diskette.
NO SYNC/WRT PRCT	Attempt to write on write-protected diskette, or else disk is bad.
WRT CRC ERR	Couldn't write on disk, disk may be bad.
RD CRC ERR	Couldn't read from disk, disk may be bad.
CANT RD NST	Drive not ready or disk is bad.
DISK/DIR FULL	Diskette is full, no more data can be stored on it. See WARNING in Chapter 7 concerning this error.
CANT RD DIR	Drive not ready or disk is bad.
CANT WRT NST	Drive not ready or disk is bad.
CANT WRT DIR	Disk may be bad.
RD ERR	Disk is bad.
WRT ERR	Disk is bad.
CANT MODIFY	Attempt to modify a file whose protect level is 3, or user didn't give permission to modify a file whose protect level is 2.
ADDR ERR	System hardware or software error.
ILLEGAL CMD	System hardware or software error.
NO DISKIO ERRS	System hardware or software error.
NO ERRS	System hardware or software error.
NOT OPEN FOR RD	System hardware or software error.
NOT OPEN FOR WRT	System hardware or software error.
NOT OPEN FOR MOD	System hardware or software error.
ALREADY OPEN	System hardware or software error.
TOO MANY FILES	System hardware or software error.
NST/DIR CONT MATCH	System hardware or software error.
PAST END OF DIR	System hardware or software error.
BAD CHNL TBL	System hardware or software error.
NO END OR DIR	System hardware or software error.
TOO MANY VOLUMES	System hardware or software error.

In a few system commands, a device name is acceptable in place of a filename. A device name is specified by an asterisk, followed by two alphabetic characters indicating a peripheral device. At present, only two device names are allowed. These are shown in Table 2-9.

Table 2-9. PDS Device Names

Name	Device
*CN	System Console
*PR	Printer

## 2.5 Printer Output

If a PDS command line has \*PR at the end of it, PDS will direct output generated by that command to the printer. This can be done with any PDS system program command. If a printer is not connected to the system, PDS will wait until one is connected. The system must be reinitialized to terminate this wait state.

Example:

```
F>C TEST1.SRC, TEST2.SRC, TEST3.SRC *PR
CREATING FILE CDS:TEST3.SRC
(This line is printed on the printer.)
```

## 2.6 System Configuration Commands

System configuration commands set and change the certain PDS system parameters. System configuration commands may be entered at the EXEC level or at any program level. Tables 2-10 and 2-11 list the commands and their parameters.

A system configuration command is invoked by typing @@ followed by a Command name and operands. The four-system configuration commands are described hereafter.

Table 2-10. System Commands

Directive	Function	Section
CONSOLE	@@ C<baud>[,<type>,<parity>[,<crdly>[,<lfldly>]]]	2.6.1
PRINTER	@@ P<baud>[,<type>,<parity>[,<crdly>[,<lfldly>[,<ffldly>[,<vtdly>]]]]]	2.6.2
TAB	@@ I[<t <sub>1</sub> >[,<t <sub>2</sub> >[,<t <sub>3</sub> >]]]	2.6.3
WIDTH	@@ W[<number of columns>]	2.6.4

Table 2-11. Operand Parameters

Command	Description
<baud>	baud rate listed in Table 2-3
<crdly>	carriage return delay in milliseconds (1-1000)
<ffldly>	form feed delay (1-1000 ms)
<lfldly>	line feed delay (1-1000ms)
<parity>	E for even parity, N for no parity
<t <sub>1</sub> >	tab column 1
<t <sub>2</sub> >	tab column 2
<t <sub>3</sub> >	tab column 3
<type>	R for RS232, C for current loop device
<number of columns>	vertical tab delay (1-1000ms)
<vtdly>	printer and console column width (10-80)

### 2.6.1 @@ CONSOLE Command

Syntax: @@ CONSOLE<baud>[,<type>,<parity>[,<crdly>[,<lfldly>]]]

The CONSOLE command sets the console parameters. The baud rate must be one of the following: 110, 150, 300, 600, 1200, 2400, 4800 or 9600. Type must be an "R" for RS232 or a "C" for current loop console. Parity must be "E" for even parity or "N" for no parity. Crdly is the carriage return delay in milliseconds. It must be a number from 0 to 1000. Lfdly is for line feed delay. It must be a number from 0 to 1000. Default parameters are RS232, no parity, zero delays. Console parameters are automatically set up when CR is typed at PDS initialization.

Example:

```
X>@@ C 1200,R,N,10,5
```

### 2.6.2 @@ PRINTER Command

Syntax: @@ PRINTER<baud>[,<type>,<parity>[,<crdly>[,<lfldly>[,<ffldly>[,<vtdly>]]]]]

The PRINTER command sets the printer parameters. Parameter description and defaults are the same as for @@ CONSOLE command, except that form feed

and vertical tab delays are added. At system initialization time, the print parameters are set to 1200 baud, RS232, no parity, zero delays.

Example:

```
X>@@ P 110,c,e,20,20,500,100
```

### 2.6.3 @@ TAB Command

Syntax: @@ TAB [<t<sub>1</sub>>[,<t<sub>2</sub>>[,<t<sub>3</sub>>]]]

The TAB command sets the tab columns for Control/I or Control/II input line control characters. Three tab columns can be set. Initial and default tabs are columns 9, 17, and 33.

Example:

```
X>@@ T 10,20,30
```

### 2.6.4 @@ WIDTH Command

Syntax: @@ WIDTH [<number of columns>]

The WIDTH command sets the printer and console column width. At system initialization this parameter is set to 72. Minimum setting is 10, maximum setting is 80.

Example:

```
X>@@ W 80
```

## 2.7 Diagnostics

Syntax: DIAGNOSTIC

The Diagnostics command, the only command in the EXEC program, causes a PDS diagnostic test to be performed. The test performs a 7-minute diagnostic of the system memory followed by a brief disk drive test. If the memory test passes, the message:

```
MEMORY TEST PASSED
```

is displayed on the console. If the test fails, a memory address is displayed on the console and servicing by National Semiconductor will be necessary. An initialized disk must be inserted in the disk drive for the disk test to succeed. The message:

```
DIAGNOSTICS PASS
```

is displayed on the console when the disk drive test passes. If the test fails, the message:

```
DISK TEST FAILED
```

is displayed.

Example:

```
X>D
DIAGNOSTICS PASS
```

The diagnostic test is also performed whenever the front panel DIAG switch is pressed. If the memory test (which takes about seven minutes) fails, the fail address will be given in the left side of the front panel display, and the test type (address, word, or bit) will be given in the right side. If this occurs, servicing by National Semiconductor is necessary. If the memory test passes, the disk test will be performed. As with the console diagnostic operation, it takes only a few seconds and requires that an initialized diskette be in the disk drive. If this test fails, DISK ERRS will be displayed on the front panel. If both tests pass, DIAG PASS will be displayed on the front panel.

# File Manager Program (FM)

## 3.1 Introduction

The File Manager program (FM) provides an interface to system disk files. FM enables the user to copy files, delete and undelete files, list the disk directory, duplicate disks, list file size and type, list space available on a disk, list and change the disk name, and perform various other functions. This chapter describes the File Manager commands and gives examples of their use.

To call FM, the user types in the @ command:

```
X>@FM
FM, REV:B
F>
```

After FM prompts for a command (F>), the user types in the necessary FM commands. These commands are summarized in Table 3-1. In commands that require a filename, if the file modifier is not specified on a filename that is to the left of TO, FM will use SRC for the default modifier. If a file modifier is not specified on a filename that is to the right of TO, FM will use the same modifier as it used for the filename to the left of TO.

## 3.2 Combine Files Command

Syntax: COMBINE <filename>,<filename>  
[,<filename>]...TO<filename>

Combines the specified disk files and saves the new disk file with the specified name. All of the disk files must be of the symbolic (SYM) file type.

Example:

```
F>C FILE1.SRC,FILE2.SRC,FILE3.SRC TO
TEST.SRC
CREATING FILE PDS:TEST.SRC
```

## 3.3 Copy File Command

Syntax: COPY<filename>TO<filename>

Copies the specified disk file to a new file on the same diskette, thus creating duplicate files with two different names.

Example:

```
F>C FILE1.SRC TO SAMPLE
CREATING FILE PDS:SAMPLE.SRC
```

## 3.4 Delete Command

Syntax: DELETE<filename>[,<filename>]...

Marks the specified files as deleted. After a file is deleted, it remains on the diskette and its name appears in the diskette directory with an asterisk beside it. It can be undeleted using the UNDELETE command. If the diskette is packed using the PACK command, the deleted file is removed from the diskette. An attempt to delete a file with a protect of 2 will cause a query to the user. The user will not be allowed to delete a file whose protect level is 3.

Example:

```
F>DE TEST.SRC, SAMPLE.MP
CANNOT DELETE FILE PDS:TEST.SRC
(protect level 3)
OK TO DELETE FILE PDS:SAMPLE.MP
(Y/N, CR= YES)? CR
```

## 3.5 Directory Command

Syntax: DIRECTORY [<option>[,<option>]...]

This command lists the diskette directory. One or both of the following options, separated by commas, may be specified.

Option A — List files in alphabetical order. Otherwise, the list is done chronologically.

Option S — A "short" listing is to be made, excluding deleted files, file IFT, version number, file#, and the number of bad, used, and available sectors on the diskette.

Example:

```
F>D A
DIRECTORY FOR: PDSUSER "PDS USER"
      FN D NAME          TYPE          SIZE  PL  VN
      2  EDIT .MP MAIN PROGRAM  20   3   1
      1  LIST .MP MAIN PROGRAM   8   3   1
SECTORS BAD:      0
SECTORS USED:    36
SECTORS FREE:   580
```

The first line in the above printout shows the diskette volume name (MASTER) and header (PDS MASTER DISKETTE).

The FN column is a chronological numbering of the first 99 undeleted files. If there are more than 99 files on the diskette, the FN field will be blank for these files.

The D (Delete) column denotes a deleted disk file with an asterisk preceding the filename.

The NAME column is an alphabetical list of the filenames and modifiers.

The TYPE column indicates the file's Internal File Type (IFT).

The SIZE column indicates the number of sectors occupied by the file.

The PL column indicates the protection level.

The VN column indicates the file version number.

The sum of the bad, used, and free sectors account for the total number of sectors on a diskette (Section 2.4). Sectors used indicates the number of sectors occupied by the files, plus a minimum of eight sectors required by PDS.

### 3.6 Duplicate File Command

Syntax: DUPLICATE<volume>:<filename>TO  
<volume>:<filename>

Copies the file from the first volume to a new file on the second volume. The volume name must be different. FM prompts the user to exchange diskettes in the disk drive as required to complete the transfer. The user must enter CR after each prompt. Control/Q instead of CR will abort the duplication.

Example:

```
F>DU VOL1:TEST.SRC TO VOL2:TEST.SRC
LOAD INDICATED VOLUME, PRESS CR
VOL1 CR
VOL2 CR
CREATING FILE VOL2:TEST.SRC
VOL1 CR
VOL2 CR
DUPLICATION COMPLETE
```

### 3.7 Duplicate Volume Command

Syntax: DUPLICATE<volume>TO<volume>

This command copies each undeleted file on the first volume to the second volume. The two volume names must be different. FM prompts the user to exchange diskettes in the disk drive as required to complete the transfer. The user must enter CR after each prompt. Control/Q instead of CR will abort the duplication. This command provides a means for making backup copies of diskettes, a recommended procedure. As many as 20 or more swaps may be needed to duplicate diskettes that have many files or large files on them. The first volume that FM will request to be loaded is the second, or destination volume. This allows a "cleaning up" operation to be performed on it prior to the duplication in order to improve the diskette's access time.

Example:

```
F>DU VOL1 TO VOL2
LOAD INDICATED VOLUME, PRESS CR
VOL2 CR
VOL1 CR
VOL2 CR
CREATING FILE VOL2:FM.MP
VOL1 CR
VOL2 CR
CREATING FILE VOL2:EDIT.MP
DUPLICATION COMPLETED
```

### 3.8 Header Command

Syntax: HEDER ["<header-string>"]

This command changes the current header to the specified header-string. If no header-string is specified, the command displays the current header but does not change it.

Example:

```
F>H "MY COP PROGRAMS"
```

### 3.9 Locate Command

Syntax: LOCATE<filename>

Lists the file type, total sectors occupied, protection level, and version number of the specified file.

Example:

```
F>L TEST.SRC
FILE TYPE: SOURCE
TOTAL SECTORS: 16
PROTECTION LEVEL: 3
VERSION NUMBER: 10
```

### 3.10 Pack File Command

Syntax: PACK<filename>

Removes all deleted files of the given name from the directory. The file can no longer be undeleted. Disk space that was occupied by the file is then freed for use by other files.

Example:

```
F>P TEST.SRC
PACKING FILE PDS:TEST.SRC (Y/N,CR = YES)? CR
```

### 3.11 Pack Volume Command

Syntax: PACK

Removes all deleted files on the diskette. The removed files can no longer be undeleted. Disk space that was occupied by the files is then freed for use by other files.

Example:

```
F>P
PACKING DISK (Y/N CR = YES)? CR
```

### 3.12 Protect Command

Syntax: PROTECT<filename>[,<plevel>]

Changes the protection level of the specified file to the protection level specified by plevel. If plevel is not specified, the command lists the protection level of the specified file but does not change it.

Example:

```
F>PR TEST.SRC,3
```

### 3.13 Rename Command

Syntax: RENAME<filename>TO<filename>

Changes the name of a file.

Example:

```
F>R TEST.SRC TO TEST.OLD
```

### 3.14 Space Command

Syntax: SPACE

Lists the number of bad, used, and available sectors on the diskette.

Example:

```
F>S
VOLUME:MASTER
SECTORS BAD: 0
SECTORS USED: 140
SECTORS FREE: 476
```

### 3.15 Undelete Command

Syntax: UNDELETE<filename>

Restores the most recently deleted version of the specified file and deletes the existing one (if any). If no deleted version exists, the following message is displayed.

NO BACKUP EXISTS

If there is more than one deleted file of the same name, the files can be successively undeleted and renamed, one at a time.

Example:

F>U TEST.SRC

### 3.16 Volume Command

Syntax: VOLUME ["<volume-name>"]

Changes the volume-name of the current diskette to the specified volume-name. If no volume-name is specified, the command lists the current volume-name but does not change it.

Example:

F>V "PDS"

Table 3-1. File Manager Command Summary

Command	Syntax	Description	Section
COMBINE FILES	C <filename>,<filename> [,<filename>]...TO<filename>	Combine symbolic files into a new file	3.2
COPY FILE	C <filename>TO<filename>	Copy file with first name to a new file with the second name.	3.3
DELETE	DE <filename>[,<filename>]	Delete files on diskette.	3.4
DIRECTORY	D [<option>[,<option>]...]	List the disk directory.	3.5
DUPLICATE FILE	DU <volume>:<filename>TO <volume>:<filename>	Copy file from one diskette to a second diskette.	3.6
DUPLICATE VOLUME	DU <volume>TO<volume>	Copy all files on one diskette to a second diskette.	3.7
HEADER	H ["<header-string>"]	List or change diskette header.	3.8
LOCATE	L <filename>	List file type, number of sectors, protection level, and version number.	3.9
PACK FILE	P <filename>	Remove deleted files of given name from the disk directory.	3.10
PACK VOLUME	P	Remove all deleted files from the disk directory.	3.11
PROTECT	PR <filename>[,<plevel>]	List or change file protection level.	3.12
RENAME	R <filename>TO<filename>	Rename file.	3.13
SPACE	S	List number of bad, used, and available sectors on the diskette.	3.14
UNDELETE	U <filename>	Undelete file.	3.15
VOLUME	V ["<volume>"]	List or change diskette volume name.	3.16

# Disk Initialization and Test (DSKIT)

## 4.1 Introduction

The Disk Initialization and Test program (DSKIT) allows the user to initialize new diskettes. Initialization consists of the following three operations:

1. Write sector sync marks on each of the disk's 616 sectors. This operation requires approximately one minute.
2. Write and verify a test pattern in each of the sectors in order to detect bad sectors. This operation requires approximately 20 minutes.
3. Write the diskette volume name and header onto the disk, and create an empty directory. This operation requires approximately 10 seconds.

These three operations can be performed with the INITIALIZE command. Although the user will probably not have use for any of the other DSKIT commands, they are described here for completeness.

To call DSKIT, type:

```
X>@DSKIT
DSKIT,REV:B
D>
```

DSKIT is then ready to accept one of the commands described in detail below. Refer to Tables 4-1, 4-2 and 4-3.

## 4.2 Initialize Command

Syntax: INITIALIZE "<volume>",<header>"

Initializes the diskette that is currently in the disk drive, giving it the specialized volume name and header string. The volume name consists of one to eight alphanumeric characters. The header string consists of one to 40 characters of any type. The system will query the user regarding initialization of the diskette before beginning the operation.

Example:

```
D>I "COPS","COP PROGRAMS"
OK TO DESTROY VOLUME "MASTER"
(Y/N, CR = YES)? N
(user forgot to put correct disk in)
D>I "COPS","COP PROGRAM"
OK TO RUN DESTRUCTIVE OPERATION ON DISK
(Y/N, CR = YES)? CR
***SECTOR MARKS COMPLETE***
***PATTERN TEST COMPLETE***
***DIRECTORY COMPLETE***
***INITIALIZATION COMPLETE***
```

Table 4-2. Command Parameter Description

Operand	Parameter
<sector>	Hexadecimal # from 0 to X'267
<track>	Hexadecimal # from 0 to X'4C
<strange>	<sector>[/<sector>]
<trrange>	<track>[/<track>]
"<volume>"	1-8 alphanumeric characters
"<header>"	1-40 characters
<aopt>*	CO, NE
<popt>*	CO, ND, NE, PA, RO, RW, WO
<topt>*	CO, PA, RO, RW, WO

\*See Table 4-3.

## 4.3 Address Test Command

Syntax: ADDRESS <strange>[<aopt>]. . .

Tests the addressing ability of the disk head. All sectors in the specified range strange are written in descending sequence with their sector addresses during Pass 1, then verified during Pass 2.

Valid Options: CO, NE

Table 4-1. DSKIT Command Summary

Command	Syntax	Description	Section
ADDRESS TEST	A<strange>[<aopt>]. . .	Tests the capability to access sectors in given range.	4-3
BAD SECTORS	B	Prints the sector numbers of bad sectors.	4-4
CLEAR	C	Clears the results of previous tests.	4-5
DIRECTORY	DI "<volume>",<header>"	Builds an empty directory.	4-6
DUMP SECTOR	D<strange>	Prints the contents of given range.	4-7
INITIALIZE	I "<volume>",<header>"	Initializes a diskette.	4-2
PATTERN TEST	P<strange>[<popt>]. . .	Tests all the sectors in given range.	4-8
SECTOR MARKS	S[<trrange>]	Writes sector marks on given track range.	4-9
STATUS 4-10	ST	Prints the drive status.	
TEST SECTOR	T<sector>[<topt>]. . .	Tests an individual sector.	4-11



Table 4-3. DSKIT Command Option Description

Option	Meaning
CO—Continuous Test	Execute specified tests (RO, WO, or RW) continuously until a console interrupt is detected.
ND—Nondestructive Test	Save original data before the test is begun, and restore data after the test has ended.
NE—No Error Message	Suppress error messages.
PA—Pattern Value	Write a specified pattern (up to four hexadecimal digits) on one or more sectors. More than one pattern may be specified.
RO—Read-Only Test	Read previously written pattern to verify the data (primarily used to test compatibility between two drives.)
RW—Read/Write Test	Write specified pattern on each sector, and read to verify (default mode).
WO—Write-Only Test	Write specified pattern on each sector but do not read.

Note: RO, RW, and WO are usually mutually exclusive, i.e., only one can be used within a given option declaration.

Example:

```
D>A 0/267
READY TO RUN DESTRUCTIVE OPERATION ON
DISK (Y/N, CR = YES)? CR
***ADDRESS TEST COMPLETE***
```

#### 4.4 Bad Sector Command

Syntax: BAD

Prints the sector numbers of all sectors found to be bad by the tests that were run after the last CLEAR, DIRECTORY, or INITIALIZE commands.

Example:

```
D>B
NO BAD SECTORS
```

#### 4.5 Clear Command

Syntax: CLEAR

Clears the results of all tests that have been executed up to this point. The command is performed automatically upon completion of the INITIALIZE DIRECTORY command.

Example:

```
D>C
```

#### 4.6 Directory Command

Syntax: DIRECTORY "<volume>","<header>"

Builds an empty directory based on all information gathered in any preceding test. This operation should be performed after any sector tests.

Example:

```
D>DI "MASTER","PDS MASTER DISKETTE"
***DIRECTORY COMPLETE***
```

#### 4.7 Dump Sector Command

Syntax: DUMP<sctrange>

Prints the contents of the specified sector range in hexadecimal with the equivalent ASCII values.

Example:

```
D>D 212/267
```

#### 4.8 Pattern Test Command

Syntax: PATTERN<strange>[<popt>]. . .

Tests all sectors in the specified range. In the normal default RW (Read/Write) Mode, each sector is written with the specified pattern, then read to verify the data. A total of five patterns may be specified with the PA option, though only one pattern may be specified during RO (Read-Only) or WO (Write-Only) tests. If the PA option is not supplied, the pattern E5E5 is assumed.

Valid Options: CO, ND, NE, PA, RO, RW, WO

Example:

```
D>P 0/267 ND PA = AAAA PA = 55555
***PATTERN TEST COMPLETE***
```

#### 4.9 Sector Marks Command

Syntax: SECTOR [<trkrange>]

Writes the sector address marks for a new diskette. This must be followed by a PATTERN TEST command over the specified range of the diskette. The final command in this initialization sequence is DIRECTORY.

Example:

```
D>S 0/4C
***SECTOR MARKS COMPLETE***
```

#### 4.10 Status Command

Syntax: STATUS

Reads sector 0 of the disk and prints the resulting disk status. The status is given as four hexadecimal digits.

The left byte is the number of errors encountered and the right byte indicates the type of error, as follows:

Right Byte	Description
X'1	No error detected
X'2	Drive not ready
X'4	Addressing error
X'8	Missing sync/write protect
X'10	Write error, CRC doesn't verify
X'20	Read error, CRC doesn't verify
X'40	Illegal disk command

Example:

```
D>ST
DISK STATUS:0001
```

#### 4.11 Test Sector Command

Syntax: TEST<sector>[<topt>]. . .

Tests a sector as in the PATTERN command. The command is normally used to test the disk drive itself rather than the actual diskette. If the PA option is not supplied, the pattern E5E5 is assumed.

Valid Options: CO, PA, RO, RW, WO

Example:

```
D>T 23A WO PA = 3333
```

# Text File Editor (EDIT)

## 5.1 Introduction

The Text File Editor (EDIT) creates and changes text files that may be subsequently used as source code for assembling programs or as documentation. A variety of commands allows the user to insert, delete, alter and list the text, and to write text to a file on floppy disk. EDIT can accept source from disk files or keyboard input. Text entered goes into the edit buffer. The edit buffer is part of the RAM reserved for system programs in the PDS system, and will hold approximately 800 lines of text. Most commands perform their operations on the contents of the edit buffer. The easiest way of editing text is using the DISK EDIT MODE. DISK EDIT MODE allows the user to specify a disk filename at the beginning of an edit and have each subsequent READ or WRITE command default refer to the specified file.

## 5.2 DISK EDIT MODE

DISK EDIT MODE is entered by using the EDIT command and specifying an edit input file that contains the source to be edited, and optionally an edit output file that will contain the source after it is edited. If an edit output file is not named, the editor will replace the edit input file with the edit output file when the disk edit mode is exited. If the edit output file is named, the edit input file will not be replaced.

Operationally, when the DISK EDIT MODE is entered, the user reads a range of lines from the edit input file to the edit buffer using an ADVANCE, READ, or POSITION command. The user performs the edits on the lines in the edit buffer, then uses another ADVANCE or POSITION command to automatically write the contents of the edit buffer to the edit output file, clear the buffer, and read the next range of lines from the edit input file. The size of the edit buffer written back to the disk need not be the same size as the block read into the buffer. When the edits are completed, the edit input file and the edit output file are closed automatically with a FINISH or TERMINATE command. To abort the DISK EDIT MODE, enter an ABORT command. In the DISK EDIT MODE, disk write errors will refer to a file called "EDIT.SYT", a temporary file for the DISK EDIT MODE.

Figure 5-1 shows the operational sequence.

Using the DISK EDIT MODE, files larger than the edit buffer can be edited. In disk edit mode, the edit buffer is treated as an "edit window". (See Figure 5-2.) The edit window (in memory) may advance through the text of the source disk file. Use of the disk edit mode, allows repositioning of large sections of text, allowing easy editing of source files much larger than the edit buffer.

A warning has been inserted when the disk has no room left for the edit. Nevertheless, care should be taken when editing to assure that there is enough room for the new edit before continuing or a disk error may occur, resulting in possible loss of a substantial portion or all of the edit.

When a write error occurs with the use of the ADVANCE, FINISH, TERMINATE, or WRITE commands, the input and output (if available) file is closed and renamed RECV (recovery). If this happens, a console message will appear. Immediately write the buffer to a different disk. Then duplicate the bad disk to a good disk. The bad disk should either be discarded or reinitialized. Now, between the original file, RECV file, and the different file, reconstruction of the edited file can be accomplished with the possible loss of only approximately 20 lines.

## 5.3 Invoking Edit

EDIT is a line editor, that is positioned by line number. Line numbers are assigned by EDIT, and are automatically adjusted when lines are inserted or deleted. EDIT is called from the disk with the @ command.

Example:

```
X>@EDIT
EDIT,REV:B
E>
```

A common sequence of operations is to call EDIT and then enter the DISK EDIT MODE.

Example:

```
X>@EDIT OLDFILE TO NEWFILE
EDIT, REV:B
AVAILABLE SECTORS: 294
INPUT FILE SECTORS: 12
E>
```

For any command which lists text, the output may be interrupted by pressing any key on the console.

## 5.4 Edit Command Mode

Edit commands are entered from the console. Text may be entered from the console or from the disk. The prompt (E>) indicates the editor is in command mode and is ready to accept a new command. (The DISK EDIT MODE is used in the command mode.)

The following command formats are listed alphabetically in Table 5-1. The definitions used in the command formats are listed in Tables 5-2. Table 5-3 is a list of EDIT Error Messages.

## 5.5 Commands Within the Edit Window (Buffer)

### 5.5.1 INSERT Command

Syntax: INSERT [TO<line>]

Accepts text from the console keyboard for insertion into the edit buffer. The text is inserted before the line indicated by the "TO line" option. If the "TO line" option is omitted, the text is appended to the end of the buffer. The prompt

line?

is given initially, and after each carriage return. The line number in the prompt is the actual number of the line about to be inserted. The insertion of lines causes

all following line numbers to be increased by a number corresponding to the number of lines inserted.

If the line number of the insert is greater than the last line of the buffer, then the text is appended to the end of the buffer. If the line number of the insert is less than the first line of the buffer, then the text is inserted in front of the first line of the buffer.

If a Control/Q is entered in column 1 in response to the EDIT command prompt, EDIT will enter the "insert mode" at the end of the buffer. If a CR is entered in column 1 in response to the edit prompt, EDIT will enter the insert mode at the current line number as if the command

I TO (current line)

had been entered. If a Control/Q or CR is entered as the first character of an inserted line, the insert mode is exited. If a Control/Q is entered in any other position, EDIT will abort that line and prompt for it again. If a CR is entered in any other position, it signifies the end of the current input line.

Examples:

1. Insert text before line 125.

```
E>I TO 125
125? $BCDADD: RC
126? _____ JP ASTART
127? $INC: SC
128? CTRL/Q# CTRL/Q is the first character
of the line, echoed on the console as a #, so
insert mode is terminated.
```

2. Insert text before the current line.

```
E>I TO
128? _____ LD
129? _____ JSR $BCDADD
130? _____ JSR $CSP CTRL/Q
130? _____ JMP $DSPLY
131? _____ JSR $MFC
132? CTRL/Q#
E>
```

3. Insert text before the current line (enter input mode)

```
E>CR
132? (ADD NEW TEXT)
133? CR (Exit input mode)
E>
```

4. Add text to the end of the buffer.

```
E>I
349? (Text also may look like this. The text in-
350? serted is just standard ASCII characters.)
351? CTRL/Q#
E>
```

5. Add more text to the end of the buffer.

```
E>CTRL/Q # Enter input mode.
351? The above command can be used to
352? insert more text.
353? CTRL/Q # Exit input mode.
```

## 5.5.2 LIST Commands

The LIST commands list text from the edit buffer. The lines are listed with their current line numbers. If the "S" (squash) option is included, the lines are left-justified, and extra blanks (more than one) are removed from between the words. The S option affects only the listing, not the text in memory.

LIST RANGE

Syntax: LIST [<range>[,<range>]...][S]

Lists a range or ranges of lines. If the range option is omitted, the entire buffer is listed beginning at the first line of the buffer. (In DISK EDIT MODE the first line of the buffer is not necessarily line 1 of the text.) "\*"PR" at the end of the command line will cause the listing to be sent to the printer.

Examples:

1. List the first line of the buffer.

```
E>L F
1 _____ .TITLE DEMO, 'SOFTWARE EXAMPLE'
E>
```

2. List lines 430/435 on the printer.

```
E>L 430/435 *PR
430 AISC 13 These lines are printed on the
printer.
431 JP $DPINC
432 JMP K # Any key (K) terminates the
listing.
```

E>

3. List the current position of the edit buffer.

```
E>L _
432 _____ JMP $DSPLY
E>
```

4. List the first line, the previous through the next line, and the last line.

```
E>L, F, P/N, L
1 _____ .TITLE DEMO, 'SOFTWARE EXAMPLE'
431 JP $DPINC
432 JMP $RST
433 ; Comment Line.
438 .END START
```

5. List with and without the S option.

```
E>L 266/271 S
266 COMP;BY SETTING ALL TO ONES
267 XAS; GET CONTENTS OF S
268 COMP;S COUNTS DOWN, SO INVERT
269 SKGBZ 0; IF KEY DOWN
270 JP $NOHOLD
271 CLRA;HOLD COUNTER

E>L 266/271
266 COMP ;BY SETTING ALL TO ONES
267 XAS ;GET CONTENTS OF S
268 COMP ;S COUNTS DOWN, SO INVERT
269 SKGBZ 0; ;IF KEY DOWN
270 JP $NOHOLD
271 CLRA ;HOLD COUNTER
E>
```

**LIST STRING**

Syntax: LIST<string>[IN<range>[,<range>]. . .][S]

Lists every line within the given range or ranges in which the specified string occurs. If no such lines exist, the message

**VOID RANGE**

is printed on the console. If the range is omitted, every occurrence of the specified string will be listed. EDIT will accept both upper- and lower-case letters. However, the user will normally use only upper-case. Within the string, upper-case characters and lower-case characters are treated as the same character. For instance:

ABC = Abc = Abc = abc = aBc = aBC = AbC

This feature is true for the LIST, ADVANCE, and POSITION commands. In all other commands that have the string option (DELETE, EDIT, CHANGE, and WRITE), the string must match exactly.

Examples:

1. List all occurrences of .WORD in lines 100 through 400.

```
E>L '.WORD' IN 100/400
283          .WORD      OFF
294      MEMORY: .WORD  03F,06,05B,04F,
                066,06D,07D,07
358      CRDRDR: .WORD  07F,067,077,07C,
                039,07E,079
E>
```

2. List all occurrences of the string RDBUF.

```
E>L "RDBUF"
VOID RANGE
E>
```

**5.5.3 NEXT Command**

Syntax: NEXT [<lines>]

Lists lines from the edit buffer. If the number of lines option is given, the listing starts at the next line and continues until the given number of lines is listed or until the end of the buffer is reached, whichever occurs first. If the number of lines option is omitted, only the next line is listed.

Examples:

1. List the next line.

```
E>N          This command is equivalent to:
103      CLRA  LN
E>
```

2. List the next five lines.

```
E>N 5
104      $NHOLD:
105          LBI      0,CNTR
106          JSR      $BCDADD
107          K #      Terminate the listing by
E>          pressing any key.
```

**5.5.4 COPY Command**

Syntax: COPY<range>[TO<line>]

Copies the specified range of lines and inserts them before the line indicated by the "TO line" option. If the "TO line" option is omitted, the copied lines are appended to the end of the buffer. The copied lines are not deleted from their original location. The buffer is renumbered after the copy.

Examples:

1. Copy lines 6 through 18 and insert them before line 23.

```
E>CO 6/18 TO 23
E>
```

2. Copy lines 100 through 120 and append them to the end of the buffer.

```
E>CO 100/120
E>
```

Note: If the editor is in DISK EDIT MODE and the buffer begins, for example, at line 110, only lines 110 through 120 are copied.

**5.5.5 DELETE Commands**

The DELETE commands delete lines of text from the edit buffer and then renumbers the buffer. If the "L" option is specified, the lines are listed on the console as they are being deleted.

Note: If the "L" option is specified, striking any key will abort the deletion of the current line and any other lines that have not been deleted already.

The specific options for the delete command are described below:

**DELete RANGE**

Syntax: DELETE<range>[,<range>]. . .[L]

Deletes the specified range or ranges of lines from the edit buffer.

Examples:

1. Delete lines 94 through 98, 101, and 103 through 105.

```
E>D 94/98,101,103/105
E>
```

2. Delete lines 203 through 206 and list the deleted lines.

```
E>D 203/206 L
203      $DOT:      XAS
204          COMP
205          SKGBZ    0
206          JP      $NOHOLD
```

**DELETE STRING**

Syntax: DELETE<string>[IN<range>[,<range>]. . .][L]

Deletes only the lines in which the specified character string occurs. If no such lines exist, the message

**VOID RANGE**

is printed on the console.

Note: Any character string found in the text must match exactly the specified character string.

## Examples:

1. Delete all lines that contain the character string RAMCLR. List all the lines.

```
E>D 'RAMCLR' L
158      JSR      $RAMCLR
170      JSR      $RAMCLR
234      JSR      $RAMCLR
282      JSR      $RAMCLR
E>
```

2. Delete all lines that contain the character string ABC from the range of line 100 through line 200.

```
E>D 'ABC' IN 100/200
VOID RANGE
E>
```

## 5.5.6 CLEAR Command

Syntax: **CLEAR**

Deletes all lines from the edit buffer.

## Example:

Clear the edit buffer and check to see if it is cleared.

```
E>CL
CLEAR CURRENT BUFFER ( Y/N, CR = YES)? CR
E>L
Buffer EMPTY List the contents of the buffer.
E>
```

## 5.5.7 MOVE Command

Syntax: **MOVE** <range>[TO<line>]

Moves a range of lines and inserts them before the line specified by the "TO line" option. The lines are deleted from their original location after the move, and the text is renumbered. If the "TO line" option is not specified, the lines are appended to the end of the buffer.

## Examples:

1. Move line 6 to the end of the edit buffer.

```
E>M 6
E>
```

2. Move lines 31 through 40 and insert them before line 68.

```
E>M 31/40 TO 68
E>
```

## 5.5.8 READ Commands

The READ commands read text from a disk file into the edit buffer. The text read is merged with any existing text in the edit buffer and the buffer is renumbered. If the buffer is filled during the course of the read, the message

```
BUFFER FULL
```

is printed on the console and the command is terminated. The buffer will contain text through the last complete line read.

The specific options for the read command are described as follows:

## READ RANGE

Syntax: **READ** [<range>] FROM <filename>  
[TO<line>]

Reads the specified range of lines from the disk file named to the edit buffer and inserts them before the line specified by the "TO line" option. If a range of lines is not specified, the active disk file will be read until an end-of-file is detected, or until the buffer is full. If the "TO line" option is omitted, the text read will be appended to the end of the buffer. The characters "F", "P", ".", "N", and "L" may not be used in the range option for this command.

## Examples:

1. Read the disk file named "UTILITY".

```
E>R FROM UTILITY This read cannot be terminated by console input.
EOF AT 246
E>
```

2. Read lines 206 through 350 from the disk file named "LIST" and insert the text before line 128.

```
E>R 206/350 FROM LIST TO 128
E>
```

3. Read from the disk file named "TEST". (The editor is in DISK EDIT MODE and using "TEST".)

```
E>R 100/200 FROM TEST
FILE ALREADY IN USE
E>
```

## READ LINES

Syntax: **READ** [<lines>]

Reads the specified number of lines from the input file and appends them to the edit buffer. If the number of lines is not specified, lines will be transferred until the edit buffer is full or until an end-of-file is reached.

NOTE: The editor must be in DISK EDIT MODE when using this command format.

## Examples:

1. Read the next 12 lines from the current disk edit input file.

```
E>R 12
E>
```

2. Read the entire file.

```
E>R
BUFFER FULL The buffer filled before the entire file was read.
E>
```

## 5.5.9 WRITE Commands

The WRITE commands write text from the edit buffer to a disk file. The specific options for the write command are described as follows:

## WRITE RANGE

Syntax: **WRITE** [<range>[,<range>]...]  
[TO<filename>]

Writes a range or ranges of lines to the disk file named by the TO <filename> option. If the range is omitted, the entire edit buffer is written to the disk file. If the TO <filename> option is omitted and the editor is in DISK EDIT MODE, the lines are appended to the current edit output file.

Examples:

1. Write the entire contents of the buffer to the disk file named "RESUME".

```
E>W TO RESUME
OK TO DELETE PDS:RESUME.SRC
(Y/N, CR = YES)? CR There was an existing
copy of the file.
CREATING FILE PDS:RESUME.SRC
E>
```

2. Write the contents of lines 1 through 200 to the disk file named "TEST1".

```
E>W 1/200 TO TEST1
CREATING FILE PDS:TEST1.SRC
E>
```

3. Write lines 152 through 393 to the current disk edit output file. (The editor is in the DISK EDIT MODE.)

```
E>W 152/393
E>
```

4. Write lines 420 through 582 to the current disk edit output file. (The editor is not in DISK EDIT MODE.)

```
E>W 420/582
NO OUTPUT FILE SPECIFIED
E>
```

5. Write the contents of the buffer to the disk file named "TEST1". (Editor is in DISK EDIT MODE and using "TEST1".)

```
E>W TO TEST1
FILE ALREADY IN USE
E>
```

#### WRITE STRING

Syntax: WRITE <string>[IN<range>[,<range>]...] [TO<filename>]

Writes all the lines within the given range or ranges of lines that contain the specified character string to the disk file named by the TO <filename> option. If the range option is omitted, all lines that contain the string are written to the disk file. If the TO <filename> option is omitted and the editor is in the DISK EDIT MODE, the lines are appended to the current edit output file.

Note: All the character strings found in the text must match the specified character string.

Examples:

1. Write all occurrences of the string XYZ to the disk file "TEST2".

```
E>W 'XYZ' TO TEST2
VOID RANGE None found.
E>
```

2. Write to the current disk edit output file all the lines, from line 168 to line 250, that contain the string "DEV02". (The editor is in DISK EDIT MODE.)

```
E>W 'DEV02' IN 168/250
E>
```

3. Write to the disk file named "TEST" all the lines that contain the string "ABCD". (The editor is not in DISK EDIT MODE.)

```
E>W 'ABCD'
NO OUTPUT FILE SPECIFIED
E>
```

#### 5.5.10 EDIT Commands

The EDIT commands allow the user to edit a range or ranges of lines. Within a line, characters may be inserted, changed, or deleted; or the line may be inserted, changed, or deleted; or the line may be extended or truncated. If the range option is omitted, the entire buffer is edited beginning at the first line. If the "S" (single) option is selected, there will be no prompt for a second edit of the same line. The control characters that may be used with this command are shown in Table 5-4. They are similar to the control characters described in Chapter 2 for the general line input routine. The line edit mode described here, however, is one of the few times when the general line input characters are not used.

#### EDIT RANGE

Syntax: EDIT [<range>[,<range>]...] [S]

Edits a range or ranges of lines.

Examples:

1. Edit line 179.

```
E>E 179
179 JRS IFBYP ;IF BYPASS
EDITS? JSR CR
179 JSR IFBYP ;IF BYPASS
EDITS? CR (in column 1 terminates the edit)
E> It is necessary to correct only as far as
the error.
```

2. Edit the buffer starting at the current line.

```
E>E .L
451 LEI 1
EDITS? CR No edits to this line.
452 AISC 9 9 should be changed.
EDITS? CTRL/Z 9 Search for the 9.
EDITS? AISC 9 Carriage stops at 9.
EDITS? AISC 5 CR Correct the line.
452 AISC 5
EDITS? CR No more edits this line.
453 LBI K# Abort the listing.
EDITS? CTRL/Q # Abort the EDIT command.
```

3. Edit lines 120 and 121 using the S option.

```
E>E 120/121S
120 LBI 0,4 With the S option, each line
EDITS? LBI 1 CR is presented for editing only
121 AISC 4 once.
EDITS? AISC 5 CR
E>
```

## EDIT STRING

Syntax: EDIT<string>[IN<range>,<range>]. . .][S]

Edits all occurrences of the specified character string within a range or ranges of lines. The character string searched for must match exactly the character string specified. For instance, to match "ABC" the editor must find "ABC". "Abc" would not match.

Example: Edit all lines which contain the string CARRY.

```
E>E 'CARRY'
104 LBI 0,CARRY  Change CARRY to CRY.
EDITS? CTRL/Z A  Search for an "A".
EDITS? LBI 0,CA  Carriage stops at A.
EDITS? LBI 0,CA CTRL/X CTRL/X CR
EDITS? LBI 0,C  Delete "AR" (" " are
echoed back.)
104 LBI 0,CRY
EDITS? CR  No more changes.
```

## 5.5.11 CHANGE Commands

The CHANGE commands change a character string or a range of columns to a specified character string throughout a range or ranges of lines. The altered lines will be displayed on the console unless the "N" (no list) option is specified. Pressing any key will abort the change for the current line and the remaining lines of the given range or ranges. The specific options for this command are described below.

## CHANGE STRING

Syntax: CHANGE<string>TO<string>[IN<range>,<range>]. . .][N]

Substitutes the second character string for the first character string throughout the specified range or ranges of lines. If no substitutions can be made, the message

VOID RANGE

is printed on the console.

For a character string in the text to be changed from the first character string specified in the command to the second character string specified in the command, it must match exactly the first character string (i.e., "ABC" does not match "abc").

Examples:

1. Change the character string ABCD to 1234 throughout the entire buffer.

```
E>C 'ABCD' TO '1234'  The editor did not find
VOID RANGE          any occurrences of the
E>                  string ABCD.
```

2. Change the character string \$3 to \$N10 in lines 100 through 200.

```
E>L '$3'  List all occurrences of the string $3.
101 JP $3
135 $3: LBI $BPI
172 JP $3
E>C '$3' TO '$N10' IN 100/200
101 JP $N10
135 $N10: LBI $BPI
172 JP $N10
E>
```

## CHANGE COLUMNS

Syntax: CHANGE<range>TO<string>[IN<range>,<range>]. . .][N]

Changes one or more columns to the specified character string in a range or ranges of lines. If crange specifies a range of columns, then the existing columns in that range are modified. If crange specifies a single column, then the specified character string is inserted starting at that column.

Examples:

1. List lines 30 through 35, then insert "\*" in column 1 in lines 30 through 35.

```
E>L 30/35
30 -----
31
32 READ INSTRUCTIONS BEFORE
33 TURNING ON PROCESSOR
34
35 -----
```

```
E>C 1 TO '*' IN 30/35
```

```
30 ** -----
31 **
32 ** READ INSTRUCTIONS BEFORE
33 ** TURNING ON PROCESSOR
34 **
35 ** -----
```

2. Change columns 2 through 3 to ";" in lines 30 through 35.

```
E>C 2/3 TO ';' IN 30/35
```

```
30 ; -----
31 ;
32 ; READ INSTRUCTIONS BEFORE
33 ; TURNING ON PROCESSOR
34 ;
35 ; -----
E>
```

This command places the contents of column 1 and deletes the contents of column 2. The remainder of the affected lines are moved one column to the left.

## 5.5.12 ALIGN Command

Syntax: ALIGN [<range>][IN<indent>][CO<crange>]

Aligns a range of lines on the columns specified by the CO crange option. If the second column number of the crange is not specified, it defaults to the width of the line. If the IN indent option is specified by indent, the first line of the range is assumed to be the start of the first paragraph.

Lines are added or deleted whenever necessary, and the text is renumbered when the ALIGN command is completed. One or more blank lines defines a paragraph.

The ALIGN command removes excess spaces within each paragraph, even from within any character string contained in the paragraph. If there are one or more spaces after the following characters before alignment, two spaces will follow each character after alignment: ".,":", "!", "?". All other characters will be followed by a single space after alignment, provided, of course,



that they were followed by at least one space before alignment.

The listing of the range of lines that were aligned may be aborted by pressing any key.

This command is used primarily for realigning documentation after text has been added or deleted. The user should be extremely cautious when using this command since all of the text within the range is aligned before any lines are listed. If incorrect numbers are given, the user could align areas he had no intention of aligning. It would be advisable to practice using this command before trying it on a large source file.

Example: Align lines 1 through 5 of the following text. Indent 5 spaces in columns 20 through 60.

```
E>_L
 1 THE FOLLOWING VERIFICATION PROCEDURE
 2 IS INTENDED TO PROVIDE THE USER WITH BOTH
 3 AN INTRODUCTION TO SYSTEM OPERATION AND
 4 A VERIFICATION OF SYSTEM SOFTWARE AND
 5 HARDWARE.
 6 THE FOLLOWING FIVE SYSTEMS WILL BE USED:
E>_AL 1/5 IN 5 CO 20/60
 1 THE FOLLOWING VERIFICATION
 2 PROCEDURE IS INTENDED TO PROVIDE THE
 3 USER WITH BOTH AN INTRODUCTION TO
 4 SYSTEM OPERATION AND A VERIFICATION
 5 OF SYSTEM SOFTWARE AND HARDWARE.
```

### 5.5.13 SCALE Command

Syntax: SCALE

Prints out a repeating string of digits starting from column 1 of the text field and continuing to column 72. This line of digits may then be compared with printed or displayed text line to determine actual column numbers.

Example:

```
E>_S
123456789-123456789-123456789-123456789-
123456789-
E>
```

## 5.6 Commands That Move the Edit Window

The ADVANCE and POSITION disk edit mode commands maintain the same line numbers as the edit input file on disk. For instance, an advance to line 100 would read lines 100 through 149 (if the size default is used). Of course, any insertions or deletions change the line numbers and the text written to the edit output file which will not, therefore, necessarily have the same line numbers as the text in the edit input file.

The ADVANCE string and the POSITION string command both have the automatic case conversion feature. That is, ABC = A**B**c = Abc, etc.

### 5.6.1 ADVANCE Commands

The ADVANCE commands advance the edit window forward only (in the direction of increasing line numbers). ADVANCE (rather than the POSITION command) normally is used to advance through a disk file. When advancing, prior to finding the first line or string, pressing any key will stop the advance and list the line the command was currently processing.

The specific options for the ADVANCE command are described below.

#### ADVANCE RANGE

Syntax: ADVANCE [<range>]

Writes the contents of the edit output file, clears the edit buffer, then copies the contents of the edit input file (starting at the next input line) to the edit output file until the lower line of the specified range is reached. Text is then read from the edit input file to the edit buffer until the upper line of the specified range is reached, or an end-of-file is reached, or the buffer is filled. If the lower line number of the specified range already has been passed (either it was in the current buffer or it previously had been written to the edit output file), the message

LINE NUMBER BEYOND RANGE

is printed on the console, and the command is aborted.

If only the lower line of a range is specified, the editor sets the upper line of the range to the lower line plus 49. For example,

A LINE

is equivalent to

A LINE/LINE + 49

Examples:

- Advance to 200.  
E> A 200 Equivalent to AD 200/249.  
E>
- Advance to 300 through 600.  
E> A 300/600  
E>
- Advance to 200.  
E> A 200  
LINE NUMBER BEYOND RANGE  
E>

#### ADVANCE STRING

Syntax: ADVANCE<string>

Writes the contents of the edit buffer to the edit output file, then copies the contents of the edit input file (starting at the next input line) to the edit output file until the specified character string is found. If the character string is found, it will be the only line written from the edit input file to the edit buffer. If the character string is not found, the contents of the edit input file are copied to the edit output file until an EOF (end-of-file) is found.

Examples:

- Advance to the first occurrence of the character string \$DEFAULT:.

```
E>A "$DEFAULT:"
1048 $DEFAULT:
E>
```

- Advance to the first occurrence of the character string ABC.

```
E>A 'ABC'
EOF AT 276 ABC was not found.
E>
```

## 5.6.2 POSITION Commands

The POSITION commands move the edit window to a new position in the edit input file. The contents of the edit buffer are written to the edit output file, the edit buffer is cleared, and then the specified lines are read into the buffer from the edit input file.

POSITION allows a user to reorganize large blocks of text in his file. For instance, in the example below, suppose a user wanted to move the sections of text designated A, B, and C so that C was the first section of text in the source file, B was next, and A was last.

	Source file before POSITION commands	Source file after POSITION commands
1	_____	_____
	A	C
500	_____	_____
501	_____	_____
	B	B
1000	_____	_____
1001	_____	_____
	C	A
1500	_____	_____

First, enter the DISK EDIT MODE, then position at the range of lines designated C, then at the range of lines designated B, then at the range of lines designated A. Finally, terminate the edit. If the source file was named "TEST", then the operation would be as follows:

```
E>E TEST Enter DISK EDIT MODE.
E>P 1001/1500 Position at section "C".
E>P 501/1000 Position at section "B".
E>P 1/500 Position at section "A".
E>T Terminate DISK EDIT MODE.
TERMINATE CURRENT EDIT (Y/N, CR = YES)? CR
OK TO DELETE FILE PDS:TEST.SRC
(Y/N, CR = YES)? CR
E>
```

The specific options for this command are described below.

### POSITION RANGE

Syntax: POSITION<range>

Positions the edit window at the specified range of lines. If the range is too large to fit into the edit buffer, the message

### BUFFER FULL

is printed on the console, and the command terminates with the last line that will fit in the buffer. If this happens, the user may use the ADVANCE command to edit the remainder of the range, then continue. (See Example 4.)

If just the first line of the range is given, the default range will be "line/line + 49."

Examples:

- Position at lines 100 through 700.

```
E>P 100/700 The range was too large.
BUFFER FULL
E>
```

- Position at lines 1 through 200.

```
E>P 1/200
E>
```

- Position at line 100. (Range will be 100/149.)

```
E>P 100
E>
```

- In this example, the user has divided his source into three sections, and plans to move section C to the beginning of the file, followed by section B, then section A (see the figure below). However, there is a problem in that the buffer is not large enough to hold section B in its entirety.

Line No.		
1	_____	_____
	A	C
500	_____	_____
501	_____	_____
	B	B
1500	_____	_____
1501	_____	_____
	C	A
2000	_____	_____

```
E>E TEST Enter DISK EDIT MODE.
E>P 1501/2000 Position at section C.
E>P 501/1500 Position at section B.
BUFFER FULL
E>LL List the last line in the buffer.
1000 JP ACOOP
E>A 1001/1500 Advance to the end of section B.
E>P 1/500 Position at section A.
E>T Terminate the edit.
TERMINATE CURRENT EDIT (Y/N, CR = YES)? CR
OK TO DELETE FILE PDS:TEST.SRC
(Y/N, CR = YES)? CR
```

### POSITION STRING

Syntax: POSITION<string>[FROM<line>]

Positions the edit window at the first line in which the specified character string occurs, beginning from the line specified by the "FROM line" option. If the "FROM line" option is not specified, the search will begin from the next input file.

If a line containing the character is found, the line is listed on the console. The edit buffer will contain only that line.

Examples:

1. Position to the first occurrence of DATA beginning from line 86.  
E>P 'DATA' FROM 86  
143 LBI 0,DATA  
E>
2. Position to the first occurrence of BLANKS.  
E>P 'BLANKS'  
683 \$GR: LBI 1,BLANKS  
E>

### 5.7 DISK EDIT MODE Setup and Quit Commands

The EDIT, FINISH, TERMINATE, and ABORT commands described in the following paragraphs allow the editor program to enter and exit DISK EDIT MODE.

#### NOTE

*The user should assure that there is ample space on his disk for the edit output file before entering DISK EDIT MODE. Upon entering the DISK EDIT MODE, the size of the available disk space and the size of the input file are displayed. If the user is creating a new file, only the available disk size is displayed.*

#### 5.7.1 EDIT Command

Syntax: EDIT<filename>[TO<filename>]

Puts the editor in DISK EDIT MODE. In the above command, the first named file is declared to be the edit input file and the second named file is declared to be the edit output file. If the edit output file does not exist, the editor will create one at a protection level equal to that of the input file. If the edit output file does exist, dialogue appropriate to its protection level will take place after the edit is completed. If a second file is not named, the editor will construct a provisional edit output file. If the edit is completed normally, the editor will delete the original edit input file and replace it with the edit output file. The protection level of the new edit input file will be the same as that of the old edit input file.

Examples:

1. Create a new edit output file.  
E>E TEST1  
CREATE NEW FILE (Y/N, CR = YES)? CR  
E>
2. Edit disk file TEST1 TO TEST2.  
E>E TEST1 TO TEST2  
E>
3. Edit disk file TEMP.A.SRC. (Editor already in DISK EDIT MODE editing TEMP.A.SRC.)  
E>E TEMP.A.SRC  
FINISH CURRENT EDIT (Y/N, CR = YES)? N  
FILE PDS:TEMP.A.SRC  
CAN'T DELETE *No permission to delete file.*  
E>

4. Edit disk file B.SRC to C.SRC. (Editor already in DISK EDIT MODE editing A.SRC.)  
E>E B.SRC TO C.SRC  
FINISH CURRENT EDIT (Y/N, CR = YES)? CR  
OK TO DELETE FILE PDS:A.SRC (Y/N, CR = YES)? CR  
E> *Now ready to begin new edit.*
5. Edit disk file B.SRC. (Editor already in DISK EDIT MODE editing A.SRC.)  
E>E B.SRC  
CONTINUE CURRENT OUTPUT FILE (Y/N, CR = YES)? CR *In this example, file A.SRC is terminated, and file B.SRC is opened.*

#### 5.7.2 FINISH Command

Syntax: FINISH

Appends the contents of the edit buffer and the remainder of the edit buffer and the remainder of the edit input file to the edit output file, terminates DISK EDIT MODE, and closes the edit input file and the edit output file. This is a normal completion.

If the editor is not in DISK EDIT MODE, this command is ignored.

Examples:

1. Finish the current edit.  
E>F  
FINISH CURRENT EDIT (Y/N, CR = YES)? CR  
OK TO DELETE FILE PDS:DIVIDE.SRC (Y/N, CR = YES)? CR  
E>
2. The editor was not in DISK EDIT MODE.  
E>F  
NOT IN DISK EDIT MODE  
E>
3. Finish the current edit.  
E>F  
FINISH CURRENT EDIT (Y/N, CR = YES)? CR  
FILE PDS:A.SRC *There was not enough space on the disk for the edit output file.*  
END OF DISK  
E>

#### 5.7.3 TERMINATE Command

Syntax: TERMINATE

Appends only the contents of the edit buffer to the edit output file, terminates the edit mode, and closes the edit input file and the edit output file. This is a normal completion.

If the editor is not in DISK EDIT MODE, this command is ignored.

Examples:

1. Terminate the current edit.  
E>I  
TERMINATE CURRENT EDIT (Y/N, CR = YES)? CR  
OK TO DELETE FILE PDS:SAMPLE.SRC (Y/N, CR = YES)? CR  
E>

2. The editor was not in DISK EDIT MODE.

```
E>I
NOT IN DISK EDIT MODE
E>
```

#### 5.7.4 ABORT Command

Syntax: ABORT

Aborts the edit mode. The edit buffer is cleared, the edit input file is closed, and the edit output file is not written. If the editor is not in the DISK EDIT MODE, this command is ignored.

Examples:

1. ABORT DISK EDIT MODE, then list the contents of the edit buffer.

```
E>AB
ABORT CURRENT EDIT (Y/N, CR = YES)? CR
OK TO DELETE FILE PDS:DIVIDE.SRC (Y/N,
CR = YES)? CR
E>L
BUFFER EMPTY
E>
```

2. The editor was not in DISK EDIT MODE.

```
E>AB
NOT IN DISK EDIT MODE
E>
```

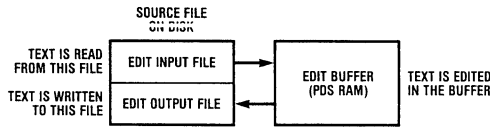


Figure 5-1. Operational Sequences of DISK EDIT MODE

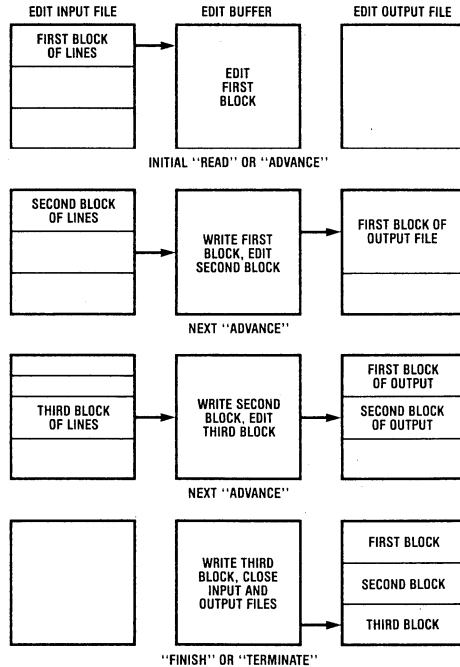


Figure 5-2. DISK EDIT MODE Edit Window Operator

**Table 5-1. Editor Commands**

The following is a list of the edit command mnemonics and formats. All commands may be abbreviated to the first two characters of the command word, and some commands may be abbreviated to the first character only. The abbreviations are indicated by an underline.

An asterisk in front of a command indicates the command is available only when the disk is inserted in the drive.

Command	Parameters	Section
*ABORT	AB	5.7.4
*ADVANCE	A [<range>]	5.6.1
ADVANCE	A<string>	5.6.1
ALIGN	AL [<range>] [IN<indent>] [CO<range>]	5.5.12
CHANGE	C<string>TO<string> [IN<range>,<range>]... [N]	5.5.11
CHANGE	C<range>TO<string> [IN<range>,<range>]... [N]	5.5.11
CLEAR	CL	5.5.6
COPY	CO<range>[TO<line>]	5.5.4
DELETE	D<range>,<range>... [L]	5.5.5
DELETE	D<string> [IN<range>,<range>]... [L]	5.5.5
EDIT	E [<range>,<range>]... [S]	5.5.10
EDIT	E<string> [IN<range>,<range>]... [S]	5.5.10
*EDIT	E<filename> [TO<filename>]	5.7.1
*FINISH	F	5.7.2
INSERT	I [TO<line>]	5.5.1
LIST	L [<range>,<range>]... [S]	5.5.2
LIST	L<string> [IN<range>,<range>]... [S]	5.5.2
MOVE	M <range> [TO<line>]	5.5.7
NEXT	N [<lines>]	5.5.3
POSITION	P <range>	5.6.2
POSITION	P <string> [FROM<line>]	5.6.2
READ	R [<range>] FROM<filename> [TO<line>]	5.5.8
*READ	R [<lines>]	5.5.8
SCALE	S	5.5.13
*TERMINATE	T	5.7.3
*WRITE	W [<range>,<range>]... [TO<filename>]	5.5.9
*WRITE	W<string> [IN<range>,<range>]... [TO<filename>]	5.5.9

Table 5-2. Command Format Definitions

Symbol/ Notation	Definition												
column	is used as a single column number in the range of 1 to 80.												
crange	(column range) is defined as: column (/column), where the first column specified indicates the beginning of a column range and the second column specified indicates the end of a column range. The default for the second column is the last column of the line. Note: In the CHANGE command, if only the first column is specified, it indicates an insert starting at the column. Note: The second column number must be equal to or greater than the first column number.												
device as	indicates an input/output device other than the disk. The legal device mnemonics are as follows: <table style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Mnemonic</th> <th>Device</th> </tr> </thead> <tbody> <tr> <td>*CN</td> <td>Console</td> </tr> <tr> <td>*PR</td> <td>Printer</td> </tr> </tbody> </table>	Mnemonic	Device	*CN	Console	*PR	Printer						
Mnemonic	Device												
*CN	Console												
*PR	Printer												
filename	indicates a legal disk filename. See Chapter 4 for a description of what constitutes a legal disk filename.												
indent	indicates the number of columns to indent the first line of each paragraph in a range of lines. (Used only in the ALiGN command.)												
line	indicates the number of a line in the edit buffer. Line may be entered as an integer in the range of 1 to 32,766 or as one of the following characters: <table style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Character</th> <th>Buffer Line Indicated</th> </tr> </thead> <tbody> <tr> <td>F</td> <td>First line</td> </tr> <tr> <td>P</td> <td>Previous line</td> </tr> <tr> <td>.</td> <td>Current line</td> </tr> <tr> <td>N</td> <td>Next line</td> </tr> <tr> <td>L</td> <td>Last line</td> </tr> </tbody> </table> <p>Note: The above characters may not be used in the READ command as part of the range specification.</p>	Character	Buffer Line Indicated	F	First line	P	Previous line	.	Current line	N	Next line	L	Last line
Character	Buffer Line Indicated												
F	First line												
P	Previous line												
.	Current line												
N	Next line												
L	Last line												
lines	indicates the number of lines to be read or the number of lines to be listed. Lines is an integer in the range 1 to 32,766.												
range	is defined as: line (/line), where the first line specified indicates the beginning line of the range, and the second line specified indicates the ending line of the range. Examples: 10/50, F/L, P, N/200, ./L, 342												
string	is a string of 0 to 15 ASCII characters enclosed in single or double quotes. If the character string contains quotes (single or double), then the quotes defining the character string must be different. Examples: "memory's" This would work. 'memory's' This would not work. ""to line"" This would work. ""to line"" This would not work.												
/	(slash) is entered as shown between the beginning and ending lines of a range or the beginning and ending columns of a range.												
[]	(brackets) indicate the enclosed item or items are optional.												
...	(ellipsis) indicates that the previous items may be repeated if desired.												

Table 5-3. Error Messages

<p style="text-align: center;"><b>ALIGN ERROR—STOP AT line number</b></p> <p>The length of the line the editor stopped at is greater than the maximum line width set or the column range specified. Either increase the line width or the column range, or make the line shorter and re-align the range.</p>
<p style="text-align: center;"><b>BUFFER EMPTY</b></p> <p>Attempted to perform action on an empty buffer.</p>
<p style="text-align: center;"><b>BUFFER FULL</b></p> <p>Attempted to exceed the protective limit of the edit buffer (i.e., next line may exceed the maximum buffer size).</p>
<p style="text-align: center;"><b>BUFFER FULL—CHANGE IGNORED</b></p> <p>The error message was caused by one of the following operations:</p> <ol style="list-style-type: none"> <li>1. "EDIT line"—buffer full after edit: changes are ignored.</li> <li>2. CHANGE command caused a buffer full (lines expanded), current line ("L") is next line to be changed.</li> </ol>
<p style="text-align: center;"><b>BUFFER FULL—STOP AT line number</b></p> <p>Buffer expanded during ALIGN command, next line to be aligned is shown.</p>
<p style="text-align: center;"><b>CANNOT DELETE OLD COPY OF OUTPUT FILE, NEW NAME:</b></p> <p>Edited output file has file of same name at protection level 3, non-deletable, user must enter new name.</p>
<p style="text-align: center;"><b>FILE ALREADY IN USE</b></p> <p>Attempted to read from or write to a file currently being used as either an input or an output file in "DISK EDIT MODE."</p>
<p style="text-align: center;"><b>FILE DOES NOT EXIST</b></p> <p>On an "EDIT filename to filename" the first filename does not exist.</p>
<p style="text-align: center;"><b>ILLEGAL COMMAND</b></p> <p>Nonexistent command used.</p>
<p style="text-align: center;"><b>ILLEGAL OPERAND</b></p> <p>The error message was caused by one of the following operations:</p> <ol style="list-style-type: none"> <li>1. Illegal operand.</li> <li>2. Disk not available and using disk-related commands ("READ FROM file, WRITE TO file," etc.).</li> </ol>
<p style="text-align: center;"><b>LINE NUMBER BEYOND RANGE</b></p> <p>In the command "ADVANCE line[/line]", the lower line of the range has already been brought into the edit buffer or written out, and therefore is not on the disk.</p>
<p style="text-align: center;"><b>NO INPUT FILE SPECIFIED</b></p> <p>Attempted to execute a "READ [line]" when not in DISK EDIT MODE and no input file specified.</p>
<p style="text-align: center;"><b>NOT IN DISK EDIT MODE</b></p> <p>The following commands are not available when not in DISK EDIT MODE: ABORT, ADVANCE, FINISH, POSITION, and TERMINATE</p>
<p style="text-align: center;"><b>NUMBER OVERFLOW</b></p> <p>The error message was caused by one of the following operations:</p> <ol style="list-style-type: none"> <li>1. The input number specified is greater than 32,766.</li> <li>2. The next input line will cause the text buffer to have a line number greater than 32,766.</li> <li>3. The range of lines to be copied will cause the text buffer to have a line number greater than 32,766.</li> </ol>

Table 5-3. Error Messages (Continued)

OUTPUT ALREADY HAS EOF
Disk error occurred when closing file. Attempted to execute ADVANCE, POSITION, READ, WRITE after a disk error on closing. Only valid commands are ABORT, FINISH, and TERMINATE.
RANGE WILL NOT FIT
The range of lines to be copied will cause the text buffer to exceed the maximum buffer size.
UNABLE TO ACCESS FILE
The editor is unable to transfer control to the file specified because an illegal character has been detected in the filename specified.
VOID RANGE
The lines referenced are not within the boundaries set by the specified ranges.

Table 5-4. Edit Command Control Characters

Control Character	Description
CTRL/A	Followed by a character string and a carriage return inserts the string after the CTRL/A. A ">" is echoed on the console for the CTRL/A.
CTRL/B	Backspace one word.
CTRL/C	Advances the carriage to the third tab setting without changing any intervening characters in the line.
CTRL/D	Truncates the rest of the line from the current carriage position.
CTRL/E	Advances the carriage to one column past the last character of the current line, provided the position of the last character is less than the width. For example, if the last character is in column 65, and the width is 72, then CTRL/E will move the carriage to column 66.
CTRL/F	Forward space one word.
CTRL/H	Backspace one character.
CTRL/I or CTRL/T	Advances the carriage to the next tab setting, changing any intervening characters in the line to spaces. Space one if past third tab.
CTRL/L	Forward space one character.
CTRL/Q	Aborts the current line modifications if entered in any column position other than column 1. If entered in column 1, CTRL/Q aborts the EDIT command and any modifications to the current line.
CTRL/W or CTRL/P	Followed by any character, advances the carriage one column beyond the next occurrence of the specified character. If there are no occurrences of the character before the carriage return, the carriage does not move.
CTRL/X	Deletes the current character and echoes a "^" in its place.
CTRL/Z or CTRL/O	Followed by any character, advances the carriage to the column containing the next occurrence of the specified character. If there are no occurrences of the character before the carriage return, the carriage does not move.
CR	(Carriage return) in column 1 terminates modifications on the current line.
K	(Any key) aborts the listing of the current line.
SHIFT/O	Backspace one character.
Underline	Backspace one character.



# COP CROSS ASSEMBLER (ASM)

## 6.1 Introduction

The COP Cross Assembler (ASM) translates symbolic program files (created with the text editor, using Assembly Language statements) into object code files (Load Modules) which contain program instructions in binary machine language format. The Load Modules, in turn, are used for loading into PDS shared-memory for debugging, for mask-programming the machine code into the appropriate COP400 device (MASKTR), or for programming test PROMs by the PDS user. The assembler also generates an output listing containing source statements with their corresponding machine code and memory locations, error messages, and other information useful to the programmer in debugging and verifying COP400 programs. Included in the listing are some warning messages with respect to emulating the COP410L/COP411L/COP420C chips with the COP400-E02 emulator.

The warnings are:

- \*1\* RAM REGISTERS ARE NOT THE SAME AS THE COP420
- \*2\* STACK ON COP410/COP411 HAS ONLY TWO LEVELS
- \*3\* "IT" INSTRUCTION VALID FOR COP420C ONLY (2-BYTE NOP ON COP402L AND COP421)
- \*4\* "HALT" INSTRUCTION VALID FOR COP420C AND COP445C ONLY (2-BYTE NOP ON COP401 AND COP402)
- \*W\* IF THE LISTING HAS BEEN SUPPRESSED, AS IN MACROS WHERE THE EXPANSION IS NOT LISTED OR BY USE OF LIST OPTIONS, THE \*W\* WILL BE PRINTED ON THE FIRST PRINTED INSTRUCTION AFTER THE LIST IS TURNED BACK ON. THIS WARNING MEANS THAT THERE WERE INSTRUCTIONS IN THE NONLISTED CODE THAT WOULD HAVE GENERATED WARNINGS HAD THE LIST BEEN ALLOWED.

The source version number is printed on the assembled program listing. The version number aids in matching current listings with different versions of the source files.

This chapter will describe the assembler statements, coding conventions, and other information necessary to use ASM.

To call ASM, the user types in the @ command:

```
X>@ASM I = <input>[,O = <output>][,L = <list>]
[,<options>]
```

```
ASM,REV:C
```

```
(Assembly now begins.)
```

```
or:
```

```
X>@ASM
```

```
ASM,REV:C
```

```
A>I = <input>[,O = <output>][,L = <list>]
```

```
<options>]
```

```
(Assembly now begins.)
```

where the assembly parameters are as follows:

Description	Definition
Input Device (required): Disk File	I = <filename>
Output Device (optional): Disk File	O = <filename>
List Device (optional): Console	L = *CN (default)
Printer	L = *PR
Disk File	L = <filename>
Listing Options (optional):	
Error Listing Only	EL
No Symbol Table List	NM
No Comment List	NC
No Listing	NL

The symbolic Assembly Language input to ASM is from a disk file created by the user. The default modifier for the input filename is SRC.

The machine code Load Module may be output to a disk file by the assembler. The default modifier for the output filename is LM.

An assembly listing may be output to the console, printer, or a disk file. The default modifier for the list filename is LST.

The Load Module and Listing will be produced only if the user specifies the "O=" or "L=" parameters, respectively.

The listing contains program assembly language statements, together with line numbers and page numbers. For assembly language statement lines which generate machine code, the hexadecimal address of memory locations and their contents are also indicated. Errors associated with assembly language statements are flagged with descriptive error messages on the appropriate statement lines. The assembler listing also produces an alphabetical listing of all symbols used in the program together with their values. Symbols which are defined but not referenced by the program are flagged with an asterisk (\*). Symbols which are referenced but undefined are flagged with a "U". The listing also indicates the number, if any, of errors encountered during the assembly, the number of ROM words (bytes) used, the source and object checksum values and the input, output, and list filenames.

Examples of invoking an assembly:

1. Assemble disk file ADD.SRC; output Load Module to disk file ADD.LM; output full listing to printer.

```
A>I = ADD,O = ADD,L = *PR CR
```

2. Assemble file DSPLY.SRC; no Load Module; output error list only to console.

```
A>I = DSPLY,L = *CN,EL CR
```

3. Assemble disk file ABC.SRC; output Load Module to disk file ABC.LM; output full listing to disk file ABC.LST:

```
A>I = ABC,O = ABC,L = ABC CR
```

4. Assemble disk file ABC.SRC, no listing.

```
A>I=ABC,NL
```

Upon pressing the carriage return key associated with each of the above commands, the assembly process will begin. The user may terminate the assembly or the output of an assembler output listing by pressing any key. The system will then interrogate the user concerning aborting the assembly as follows:

```
CONTINUE ASSEMBLY (Y/N, CR = YES)?
```

Pressing "N CR" will abort the assembly, terminating the printing of an assembly output listing if in progress. Pressing "Y CR" or "CR" will result in a continuation of the assembly.

The disk containing ASM must be loaded into the disk drive prior to calling the assembler via the @ASM command. This may be the PDS master diskette or another disk to which ASM has been copied using the File Manager (see Chapter 3). After calling the Assembler, the user must insert the disk containing the source code file to be assembled into the drive disk. (If the file to be assembled is contained on the same disk as the ASM disk, then no change of disks is required.)

If the user program to be assembled is a disk file which resides on the same disk as ASM, the user may call and invoke an assembly after loading the disk containing both programs by combining the call of the Assembler program and Assembler parameter specifications into one command. The following is an illustration of this technique. Note that a space must be inserted between the Assembler call (@ASM) and Assembler parameter specifications.

Example:

To call the Assembler and begin an assembly of file ADD.SRC (as in Example #1) contained on the same disk. enter:

```
@ASM I=ADD,O=ADD,L=*PR CR
ASM,REV:C
```

(Assembly of ADD.SRC now begins.)

## 6.2 The Assembly Process

If an assembler were not available, programs would have to be written in machine code. The binary code for each instruction would have to be determined and manually entered into the machine. Transfer-of-control instructions, such as JMP, would require tedious manual calculation of the JMP address to allow calculation of the machine code. Instructions with operands, such as AISC, would require manual insertion of the operand value into the machine code.

An assembler simplifies the programmer's task in several ways:

1. Each instruction is represented by an instruction mnemonic instead of the less intelligible binary machine code. The assembler translates the mnemonic into the appropriate code. For example, the COP400 No-Operation instruction is represented by the mnemonic "NOP". The assembler translates this into the code 01000100.

2. Instructions which are to be referenced by transfer-of-control instructions may be labeled with a label. (See Section 6.3, Label Field, for parameters.) The label consists of one to six alphanumeric characters followed by a colon (:). The label precedes the mnemonic of the instruction it labels. For example, the label CLEAR: precedes the mnemonic CLRA in the following instruction:

```
CLEAR: CLRA
```

A transfer instruction may specify the label to pass control. The assembler assigns the appropriate address to the label, and then uses the address to determine the proper machine code for the transfer instruction. For example, if the above CLRA instruction is at address 3A7, and it is desired to jump to this instruction from elsewhere in the program, the jump instruction

```
JMP CLEAR
```

may be used rather than JMP 3A7. The assembler calculates the appropriate address (3A7).

3. Instructions with operands may be written with the operand following the mnemonic. The assembler will insert the value of the operand into the machine code. For example, AISC 7 is translated by the assembler into the code:

```
0 1 0 1      0 1 1 1
  AISC          7
```

The above three functions are present in almost every assembler. The COP assembler has several other special features which further ease the programmer's task:

4. Values may be assigned to "English-like" words, called symbols, and these symbols may be used as the operands for instructions. For example, the value 3 may be assigned to the symbol COUNT:

```
COUNT = 3
```

and this symbol may be used as an operand for instructions:

```
LBI COUNT (Equivalent to LBI 3.)
```

This feature is often used when a value may be changed during the process of program development. In this example, only the value assigned to COUNT needs to be changed. If COUNT was not used, all LBI 3 instructions throughout the entire program would have to be changed.

5. An operand may consist of an arithmetic expression. The expression will be evaluated by the assembler and its value for the operand.

Examples:

- a. LBI COUNT + 1
- b. STII - 2
- c. JMP . + 3
- d. AISC 3\*SIZE-LEN/2

Expressions may be used to improve clarity and to simplify alterations of the program by assigning values to symbols at the front of the program, and using them in arithmetic expressions for

instruction operands. Another use of an expression is shown in 5c above, which jumps to the current instruction plus 3, thus precluding the necessity for a label on this instruction.

6. Special assembler statements called directives give the user further flexibility in writing programs. Directives are available to assign a title to the program, specify the COP400 chip number and options, specify program page numbers, feed pages and lines of the output listing, perform conditional assembly of instructions, etc.
7. Assembler procedures, or MACROS, allow the programmer to give an "English-like" name, called the MACRO NAME, to sequences of instructions that are frequently used, and to insert these instructions into the program simply by stating the MACRO NAME.

The Assembler performs its functions by reading through the Assembly Language statements sequentially from top to bottom, generating the machine code and a program listing as it proceeds. Since it reads statements sequentially, a special problem occurs which must be overcome. Specifically, suppose the Assembler encounters the statement

```
JMP CLEAR
```

but has not yet encountered the label CLEAR. It will be unable to generate machine code for the instruction. This problem is solved by making the assembler perform two "passes" through Assembly Language statements.

Pass 1 of the assembler does not generate a Load Module or a Listing. Its purpose is to assign address values to labels. It does this by using an internal counter called a "location counter." The location counter is initialized to zero at the beginning of each pass. Each time the assembler encounters a single-byte COP400 instruction, the location counter is incremented by one. Each time the assembler encounters a double-byte COP400 instruction, the location counter is incremented by two. The location counter thus keeps track of the ROM address of the next COP400 instruction. In this respect it is similar to a COP400 program counter (PC) register. As the assembler encounters program labels, the labels are assigned the current value of the location counter. In this way, the assembler builds a table of label values which can be used during pass 2 to generate machine code for transfer of control instructions.

Pass 2 of the assembler generates the Load Module and/or Listing, as specified by the user. It uses the table of label values generated during pass 1 to calculate machine code values for transfer of control instructions. It also uses the location counter to determine the address which each COP400 instruction should occupy. The Load Module contains this address information.

The user may alter the value of the location counter with special Assembly Language statements (described later). Care must be exercised when doing this so as not to try to put two different COP instructions in the same ROM location.

### 6.3 Introduction to Assembly Language Statements

The input to the assembler consists of a sequence of Assembly Language statements. There are three types of Assembly Language statements:

1. Instruction statements, which provide a COP400 instruction mnemonic to be translated by the assembler.
2. Directive statements which provide the assembler with information or request it to perform specific tasks.
3. Assignment statements, which assign values to symbols.

Each statement is written using the following characters:

Letters—A through Z

Numbers—0 through 9

Special Characters—! \$ % ' ( ) \* + , - / ; : < = > b

Note: "b" indicates a blank.

These statements are entered into the assembler input file using the text editor (Chapter 5), and following certain coding conventions. Each statement contains from one to four fields in the following order:

label field operation field operand field comment field

Since the assembler accepts free-form statements, the user may disregard specific field boundaries, provided the appropriate delimiters for each field are used. However, for clarity and readability, the use of field boundaries is highly recommended. Useful boundaries can be achieved with the PDS control I or T tab function described in Section 2.7. PDS initially sets tabs at columns 9, 17, and 33. The @@TAB command described in Section 2.7 can be used to change these settings if desired. The command field may extend to column 72. Following is a description of each field.

#### LABEL FIELD

The label field is optional and may contain a symbol used to identify a statement referenced by other statements. When the assembler encounters a label, it assigns it to the current value of the location counter. More than one label may appear in the label field, in which case any of the labels may be used to reference the labeled location. A label may appear by itself in a statement, in which case it refers to the next instruction or data word in the source program. A colon (;) must be used to delimit (terminate) each label.

Labels are the most common means of referencing address locations.

Example:

```
JMP SUB
```

```
.
```

```
.
```

```
.
```

```
SUB: CLRA
```

A label must conform to the following rules:

1. A label may contain one or more alphanumeric characters, the first of which must be either a letter or a dollar sign (\$). Although up to 32 characters may be included, only the first six characters are recognized by the assembler program. Therefore, the programmer must ensure that a long label is unique in the first six characters.

Example:

```

LONGLA
LONGLABEL1 } are identical to the assembler
LONGLABEL2 }
    
```

2. If the first character in the label is a dollar sign (\$), the label is defined as a local label. The .LOCAL directive allows the programmer to specify that local labels appearing between two .LOCAL directives are accessible only within that region of the program (see Section 6.4.3). This enables the programmer to use identical labels throughout a program without causing a conflict between label names. Within a local region, a local label must be unique in the first four alphanumeric characters, not including the dollar sign (\$).

Example:

```

$ABCD
$ABCDEF } are identical labels to the
           } assembler
    
```

3. No special characters or embedded blanks may appear within a label.
4. A label represents a memory address and, hence, must have a value ranging between 0 and the maximum ROM address of the COP400 chip being used.

Several examples of labels follow:

Legal Labels	Illegal Labels	Reason Illegal
\$ABC	LONGLABEL1	First six characters are not unique
LONGLA	LONGLABEL2	
AB2	2AB	First character must be a letter or a dollar sign
\$2	2CDE	
XYZ	XYZ\$	Last character is not alphanumeric
\$ABCDEF	\$ABCDE	First four characters of the local labels are not unique
\$ABC2EF	\$ABCDF	

A label referencing an instruction need not be on the same line as the instruction — the label will be assigned the value of the address of the first instruction location following the label. This allows the programmer, when writing source code, to devote a separate line with comments to labels, providing clearer documentation of the program and allowing for easier editing of the source code. (An edit of a "label-line" instruction often involves a change of the label location.)

Example:

```

SUB:                ;SUBTRACT ROUTINE
CLRA                ;CLEAR ACCUMULATOR
    
```

Note: The label "SUB" will be assigned the value of the address of the CLRA instruction.

The label field may also contain a symbol, without a following colon (:). This format is used for the assignment statement (Section 6.4.2).

### OPERATION FIELD

The operation field is mandatory and contains an identifier indicating which type of statement it is.

In an instruction statement, the operation field contains the mnemonic name of the desired instruction. For example:

```

label  operation
SUB:   CLRA
    
```

Valid COP400 instruction mnemonics are provided in Table 6-2. The operation field of an instruction statement is often called a mnemonic field.

In a directive statement, the operation field contains a period (.) immediately followed by the name of the desired directive. For example:

```
.END
```

Valid directive names are provided in Table 6-6.

In an assignment statement, the operation field contains an equal sign (=). (See Section 6.4.2.)

One or more blanks terminate the operation field.

### OPERAND FIELD

The operand field contains entries that identify data to be acted upon by the operation defined in the operation field. Many statements do not require use of the operand field. For those that do, the operand field usually consists of one or two expressions, separated by a comma.

An expression is composed of terms. There are seven types of terms:

1. A decimal constant is a decimal number that does not begin with zero. Leading zeros for decimal data are not permitted, except for the simple case of the constant 0.  
Examples: 3, 234, -10.
2. A hexadecimal constant term is a hexadecimal number that starts with "X" or with a leading zero.  
Examples: X'23A, 07B, X'F.
3. A string constant term is a single character enclosed in single quote marks.  
Examples: 'Z', '\$', '3'.  
To use a single quote mark for a string constant, write four quotemarks: ""'"".
4. A label term is described above under the label field description.
5. A symbol term is constructed in the same way as a label term, but is used differently. (See Section 6.4.2.)
6. The location counter term is a single dot (.). The dot represents the location counter, and, if it appears within an expression, it is replaced by the current value of the location counter.  
Example: JMP .+2

- A lower-half term is represented by L (term). An upper-half term is represented by H (term). When the assembler encounters one of these in an expression, it replaced it with either the lower or the upper eight bits of the value of the symbol, respectively.

Examples:

H(X'172F) is replaced by X'17

L(X'172F) is replaced by X'2F

H(X'00FF) is replaced by X'00

L(X'00FF) is replaced by X'FF

Terms are represented internally in the assembler in 16-bit binary notation. Negative numbers are represented by two's complement notation. In this notation, the negative of a number is formed by complementing each bit in the data word and adding one to the complemented number. The sign of the number is indicated by the most significant bit. When the most significant bit is 0, the number is positive or zero; when the most significant bit is 1, the number is negative. The maximum range for a 16-bit number in this format is  $7FFF_{16}$  (+32767<sub>16</sub>) to  $8000_{16}$  (-32768<sub>10</sub>).

String constants are represented internally by the appropriate 8-bit ASCII code.

An expression may consist of a single term.

Examples:

5

H'3C

'Q'

SUB

H(H'3CF)

L(SUB)

Alternatively, an expression may consist of two or more terms combined using the operators shown in Table 6-1.

Examples:

36 + SUB

X'3F0 - 10

X'7F&'Q'

3\*5!XYZ

%SUB/2

The multiterm expression is evaluated by the assembler program in a left-to-right order regardless of the operators used between the terms. However, parentheses are permitted for the purpose of grouping the terms of a multiterm expression. They may be nested up to nine-deep within a multiterm expression, with the innermost parenthetical operation being resolved first.

The constructs "A<B", "A=B", and "A>B" cause the specified comparison to be made. The result is 1 if the comparison is true and 0 if the condition is false.

Example:

$I = ((2 + 3 * (4 + 5)) / 6$

$L(TABLE) + X'10$

$100 - 1$

$ENTRY1 + ENTRY2 - 4$

$A > B * DISKAD$

$(100 - 1 * 12) + H(X'300)$

MULTITERM EXPRESSIONS

The magnitude of the expression must be compatible with the memory storage available for the expression. For example, if the expression is to be stored in an 8-bit memory word, then the value of the expression must not exceed X'FF.

Example:

JMP X'40 + CHAR *Expression value must not exceed X'3FF for COP420 (1024 bytes of program memory).*

If the expression is used in conjunction with the JP instruction to transfer control to a new ROM word on the same page, then the value must not exceed  $N * 40_{16} + 3E_{16}$  or precede  $N * 40_{16}$  where N = the number of the current page.

Example:

.PAGE 0

.

.

.

JP TABLE+4

Expression value must not exceed  $3E_{16}$  or precede 0.

Some statements consist of a mandatory first expression and an optional second expression. When such a statement is encountered by the assembler and the operand field contains two expressions, the assembler will left shift the value of the left expression to four bits, and will then add to it the value of the right expression, which must evaluate to less than  $16_{10}$  (four bits). This feature is useful on the LBI instruction.

Example: LBI 3,15

In this example, the assembler evaluates the left expression (3), shifts it left four bits to obtain the value X'30, then evaluates the right expression (15), and finally adds it to X'30, obtaining a result of X'3F. This value is then used to determine the correct machine code for the LBI instruction. The above example is thus equivalent to:

LBI X'3F

Table 6-1. ASM Arithmetic and Logical Operators

Operator	Function	Type
+	Addition	Binary
-	Subtraction	Unary or Binary
*	Multiplication	Binary
/	Division	Binary
%	Logical NOT	Unary
&	Logical AND	Binary
!	Logical OR	Binary
<	"Less Than"	Binary
=	"Equal To"	Binary
>	"Greater Than"	Binary

## COMMENT FIELD

Comments are optional descriptive notes which are printed on the assembler output listing for programmer reference and documentation. Comments should be included throughout the program to explain sub-routine linkages, data formats, algorithms used, formats of inputs processed, and so forth. A comment may follow a statement on the same line, or the comment may be entered on one or more separate statement lines. The comment has no effect on the assembled Load Module (.LM) file.

The following conventions apply to comments:

1. A comment must be preceded by a semicolon (;).
2. All ASCII characters, including blanks, may be used in comments.
3. Comments should not extend beyond column 72, but a comment may be carried over on the following line (preceded by a semicolon).

Note: When listing a COP400 program on the system printer, comments are listed to column 63 only.

Example:

```

Label      Operand
GETVAL: JSR SAVREG ;LOAD MEMORY DATA
                INTO A

```

The label, GETVAL, is a label name for the address of this instruction. Thus, GETVAL can be used in other statements (preceding or following) to reference this statement. The instruction mnemonic JSR specifies the COP400 instruction. The operand field for the JSR instruction is the symbol SAVREG. The comment field is separated from the operand field by a semicolon (;). Spaces on each side of the semicolon are optional. The comment allows the programmer to quickly identify the operation performed by the instruction.

## 6.4 Assembler Statements

The following sections describe the COP Assembly Language statements in detail. Some statements have optional fields. These optional fields will be enclosed in brackets ([]) to indicate that they are optional.

### 6.4.1 Instruction Statements

There are approximately 60 COP400 instructions, all of which are applicable to the COP440. The COP420 instruction set is a subset of the COP440 instruction set. The COP410 instruction set, is a subset of the COP420 instruction set (COP410 instruction set < COP420 instruction set < COP440 instruction set). Also, the COP421 and COP411, which lack specific inputs, cannot use some instructions that are present in their related COP devices, the COP420 and COP410, respectively. Refer to the MOS Databook or the specific Data Sheet for information on the instruction set of the particular COP device which the assembler code is being written for.

COP400 Series Assembler instruction statements fall within one of the following six classes:

- Arithmetic Instructions
- Transfer-of-Control Instructions
- Memory Reference Instructions
- Register Reference Instructions
- Test Instructions
- Input/Output Instructions

Table 6-2 contains a summary of the COP400 series instruction set, grouped according to one of the above six classes. Additional instructions which will be included in the COP400 instruction set are not indicated. (Refer to COP440 data sheet.) This table provides the assembly mnemonic and operand, hexadecimal code, machine code (binary), data flow, skip conditions and description for each instruction. Refer to Table 6-3 for definitions of symbols used in describing the COP400 series instruction set. The Notes to Table 6-2 provide additional information to assist the user in understanding the operations of specific instructions. For further detailed information on the nature and use of the COP400 series instruction set, examples of assembly language routines and programming techniques, information on the electrical specifications and architecture of each COP400 series device, see the MOS Data Book or the specific Data Sheet. Refer to Table 6-4 for an alphabetical listing of all COP400 series instructions showing their hexadecimal opcode. Also refer to Table 6-5 for a hexadecimal opcode ordered list of the COP400 series instruction set. These latter two tables do not include references to the additional COP400 instructions.

### 6.4.2 Assignment Statements

```

symbol = (<expression> [<expression>]
[;<comments>]

```

The Assignment Statement assigns the value of the expression on the right of the equals sign to the symbol on the left of the equals sign. If two expressions are given, the value of the leftmost is shifted left by four bits, and the rightmost expression, which must evaluate to less than  $16_{10}$ , is added to this value. The Assignment Statement does not generate machine code. It simply assigns a value to a symbol. When the symbol is used in a COP instruction statement operand field, the assigned value is used to generate code.

Examples:

```

X = 3,15 ;ASSIGN VALUE OF X'3F TO
        ;"X"
LBI X ;GENERATE LBI 3,15
        ;INSTRUCTION CODE
Y = 5 ;ASSIGN VALUE OF 5 TO "Y"
AISC Y ;GENERATE AISC 5
        ;INSTRUCTION CODE

```



Table 6-2. COP400 Instruction Set

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
ARITHMETIC INSTRUCTIONS						
ASC		30	$\boxed{0011 0000}$	$A + C + \text{RAM}(B) \rightarrow A$ Carry $\rightarrow C$	Carry	Add with Carry, Skip on Carry
ADD		31	$\boxed{0011 0001}$	$A + \text{RAM}(B) \rightarrow A$	None	Add A to RAM
ADT		4A	$\boxed{0100 1010}$	$A + 10_{10} \rightarrow A$	None	Add Ten to A
AISC	y	5-	$\boxed{0101 y}$	$A + y \rightarrow A$	Carry	Add Immediate, Skip on Carry (y $\neq$ 0)
CASC		10	$\boxed{0001 0000}$	$\bar{A} + \text{RAM}(B) + C \rightarrow A$ Carry $\rightarrow A$	Carry	Complement and Add with Carry, Skip on Carry
CLRA		00	$\boxed{0000 0000}$	$0 \rightarrow A$	None	Clear A
COMP		40	$\boxed{0100 0000}$	$\bar{A} \rightarrow A$	None	Ones complement of A to A
NOP		44	$\boxed{0100 1000}$	None	None	No Operation
RC		32	$\boxed{0011 0010}$	"0" $\rightarrow C$	None	Reset C
SC		22	$\boxed{0010 0010}$	"1" $\rightarrow C$	None	Set C
XOR		02	$\boxed{0000 0010}$	$A \oplus \text{RAM}(B) \rightarrow A$	None	Exclusive-OR A with RAM
TRANSFER OF CONTROL INSTRUCTIONS						
JID		FF	$\boxed{1111 1111}$	ROM (PC <sub>9:8</sub> ,A,M) $\rightarrow$ PC <sub>7:0</sub>	None	Jump Indirect (Note 3)
JMP	a	6- --	$\boxed{0110 00 a_{9:8}}$ $\boxed{a_{7:0}}$	$a \rightarrow \text{PC}$	None	• Jump
JP	a	--	$\boxed{1 a_{6:0}}$ (pages 2,3 only) or $\boxed{11 a_{5:0}}$ (all other pages)	$a \rightarrow \text{PC}_{6:0}$  $a \rightarrow \text{PC}_{5:0}$	None	Jump within Page (Note 4)
JSRP	a	--	$\boxed{10 a_{5:0}}$	PC+1 $\rightarrow$ SA $\rightarrow$ SB $\rightarrow$ SC  0010 $\rightarrow$ PC <sub>9:6</sub> a $\rightarrow$ PC <sub>5:0</sub>	None	Jump to Subroutine Page (Note 5)
JSR	a	6- --	$\boxed{0110 10 a_{9:8}}$ $\boxed{a_{7:0}}$	PC+1 $\rightarrow$ SA $\rightarrow$ SB $\rightarrow$ a $\rightarrow$ PC	None	• Jump to Subroutine
RET		48	$\boxed{0100 1000}$	SC $\rightarrow$ SB $\rightarrow$ SA $\rightarrow$ PC	None	Return from Subroutine
RETSK		49	$\boxed{0100 1001}$	SC $\rightarrow$ SB $\rightarrow$ SA $\rightarrow$ PC	Always Skip on Return	Return from Subroutine then Skip



Table 6-2. COP400 Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>MEMORY REFERENCE INSTRUCTIONS</b>						
CAMQ		33	<u>0 0 1 1   0 0 1 1</u>	A → Q <sub>7:4</sub>	None	Copy A, RAM to Q
		3C	<u>0 0 1 1   1 1 0 0</u>	RAM(B) → Q <sub>3:0</sub>		
CQMA		33	<u>0 0 1 1   0 0 0 1</u>	Q <sub>7:4</sub> → RAM(B)	None	Copy Q to RAM, A
		2C	<u>0 0 1 0   1 1 0 0</u>	Q <sub>3:0</sub> → A		
LD	r	-5	<u>0 0   r   0 1 0 1</u>	RAM(B) → A Br ⊕ r → Br	None	Load RAM into A, Exclusive-OR Br with r
LDD	r,d	23	<u>0 0 1 0   0 0 1 1</u>	RAM(r,d) → A	None	Load A with RAM pointed to directly by r,d
		--	<u>0 0   r   d</u>			
LQID Indirect		BF	<u>1 0 1 1   1 1 1 1</u>	ROM(PC <sub>9:8</sub> ,A,M) → Q SB → SC	None	Load Q (Note 3)
RMB	0	4C	<u>0 1 0 0   1 1 0 0</u>	0 → RAM(B) <sub>0</sub>	None	Reset RAM Bit
	1	45	<u>0 1 0 0   0 1 0 1</u>	0 → RAM(B) <sub>1</sub>		
	2	42	<u>0 1 0 0   0 0 1 0</u>	0 → RAM(B) <sub>2</sub>		
	3	43	<u>0 1 0 0   0 0 1 1</u>	0 → RAM(B) <sub>3</sub>		
SMB	0	4D	<u>0 1 0 0   1 1 0 1</u>	1 → RAM(B) <sub>0</sub>	None	Set RAM Bit
	1	47	<u>0 1 0 0   0 1 1 1</u>	1 → RAM(B) <sub>1</sub>		
	2	46	<u>0 1 0 0   0 1 1 0</u>	1 → RAM(B) <sub>2</sub>		
	3	4B	<u>0 1 0 0   1 0 1 1</u>	1 → RAM(B) <sub>3</sub>		
STII	y	7-	<u>0 1 1 1   y</u>	y → RAM(B) Bd + 1 → Bd	None	Store Memory Immediate and Increment Bd
X	r	-6	<u>0 0   r   0 1 1 0</u>	RAM(B) ↔ A Br ⊕ r → Br	None	Exchange RAM with A, Exclusive-OR Br with r
XAD	r,d	23	<u>0 0 1 0   0 0 1 1</u>	RAM(r,d) ↔ A	None	Exchange RAM with A pointed to directly by r,d
		--	<u>1 0   r   d</u>			
XDS	r	-7	<u>0 0   r   0 1 1 1</u>	RAM(B) ↔ A Bd - 1 → Bd Br ⊕ r → Br	Bd decrements past 0	Exchange RAM with A and Decrement Bd, Exclusive-OR Br with r
XIS	r	-4	<u>0 0   r   0 1 0 1</u>	RAM(B) ↔ A Bd + 1 → Bd Br ⊕ r → Br	Bd increments past 15	Exchange RAM with A and Increment Bd, Exclusive-OR Br with r
<b>REGISTER REFERENCE INSTRUCTIONS</b>						
CAB		50	<u>0 1 0 1   0 0 0 0</u>	A → Bd	None	Copy A to Bd
CBA		4E	<u>0 1 0 0   1 1 1 0</u>	Bd → A	None	Copy Bd to A
LBI	r,d	--	<u>0 0   r   (d-1)</u> (d = 0, 9:15) or	r,d → B	Skip until not a LBI	Load B Immediate with r,d (Note 6)
		33	<u>0 0 1 1   0 0 1 1</u>			
		--	<u>1 0   r   d</u> (any d)			
LEI	y	33	<u>0 0 1 1   0 0 1 1</u>	y → EN	None	Load EN Immediate (Note 7)
		6-	<u>0 1 1 0   y</u>			
XABR		12	<u>0 0 0 1   0 0 1 0</u>	A ↔ Br (0,0 → A <sub>3</sub> ,A <sub>2</sub> )	None	Exchange A with Br

Table 6-2. COP400 Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
TEST INSTRUCTIONS						
SKC		20	0010 0000		C = "1"	Skip if C is True
SKE		21	0010 0001		A = RAM(B)	Skip if A Equals RAM
SKGZ		33	0011 0011		G <sub>3:0</sub> = 0	Skip if G is Zero (all 4 bits)
		21	0010 0001			
SKGBZ		33	0011 0011	1st byte	G <sub>0</sub> = 0 G <sub>1</sub> = 0 G <sub>2</sub> = 0 G <sub>3</sub> = 0	Skip if G Bit is Zero
	0	01	0000 0001			
	1	11	0001 0001	2nd byte		
	2	03	0000 0011			
	3	13	0001 0011			
SKMBZ	0	01	0000 0001		RAM(B) <sub>0</sub> = 0	Skip if RAM Bit is Zero
	1	11	0001 0001		HAM(B) <sub>1</sub> = u	
	2	03	0000 0011		RAM(B) <sub>2</sub> = 0	
	3	13	0001 0011		RAM(B) <sub>3</sub> = 0	
SKT		41	0100 0001		A time-base counter carry has occurred since last test	Skip on Timer (Note 3)

INPUT/OUTPUT INSTRUCTIONS

ING		33	0011 0011	G → A	None	Input G Ports to A
		2A	0010 1010			
ININ		33	0011 0011	IN → A	None	Input IN Inputs to A (Note 2)
		28	0010 1000			
INIL		33	0011 0011	IL <sub>3</sub> , "1", "0", IL <sub>0</sub> → A	None	Input IL Latches to A (Notes 2 and 3)
		29	0010 1001			
INL		33	0011 0011	L <sub>7:4</sub> → RAM(B)	None	Input L Ports to RAM, A
		2E	0010 1110	L <sub>3:0</sub> → A		
OBD		33	0011 0011	Bd → D	None	Output Bd to D Outputs
		3E	0011 1110			
OGI	y	33	0011 0011	y → G	None	Output to G Ports Immediate
		5-	0101  y			
OMG		33	0011 0011	RAM(B) → G	None	Output RAM to G Ports
		3A	0011 1010			
XAS		4F	0100 1111	A ↔ SIO, C → SK	None	Exchange A with SIO (Note 3)

**Note 1:** All subscripts for alphabetical symbols indicate bit numbers unless explicitly defined (e.g., Br and Bd are explicitly defined). Bits are numbered 0 to N where 0 signifies the least significant (low-order, right-most) bit. For example, A<sub>3</sub> indicates the most significant (left-most) bit of the 4-bit A register.

**Note 2:** The ININ and INIL instructions are not available on the 24-pin COP421, since this device does not contain the IN inputs.

**Note 3:** For additional information on the operation of the XAS, JID, LQID, INIL, and SKT instructions, see data sheet.

**Note 4:** The JP instruction allows a jump, while in subroutine pages 2 or 3, to any ROM location within the two-page boundary of pages 2 or 3. The JP instruction, otherwise, permits a jump to a ROM location within the current 64-word page. JP may not jump to the last word of a page.

**Note 5:** A JSRP transfers program control to subroutine page 2 (0010 is loaded into the upper 4 bits of P). A JSRP may not be used when in pages 2 or 3. JSRP may not jump to the last word in page 2.

**Note 6:** LBI is a single-byte instruction if d = 0, 9, 10, 11, 12, 13, 14, or 15. The machine code for the lower 4 bits equals the binary value of the "d" data minus 1, e.g., to load the lower four bits of B (Bd) with the value 9 (1001<sub>2</sub>), the lower 4 bits of the LBI instruction equal 8 (1000<sub>2</sub>). To load 0, the lower 4 bits of the LBI instruction should equal 15 (1111<sub>2</sub>).

**Note 7:** Machine code for operand field y for LEI instruction should equal the binary value to be latched into EN, where a "1" or "0" in each bit of EN corresponds with the selection or deselection of a particular function associated with each bit. (See Functional Description, EN Register.)

**Table 6-3. COP400 Instruction Set Table Symbols**

Symbol	Definition
<b>INTERNAL ARCHITECTURE SYMBOLS</b>	
A	4-bit Accumulator
B	6-bit RAM Address Register
Br	Upper 2 bits of B (register address)
Bd	Lower 4 bits of B (digit address)
C	1-bit Carry Register
D	4-bit Data Output Port
EN	4-bit Enable Register
G	4-bit Register to latch data for G I/O Port
IL	Two 1-bit Latches associated with the IN <sub>3</sub> or IN <sub>0</sub> Inputs
IN	4-bit Input Port
L	8-bit TRI-STATE I/O Port
M	4-bit contents of RAM Memory pointed to by B Register
PC	10-bit ROM Address Register (program counter)
Q	8-bit Register to latch data for L I/O Port
SA	10-bit Subroutine Save Register A
SB	10-bit Subroutine Save Register B
SC	10-bit Subroutine Save Register C
SIO	4-bit Shift Register and Counter
SK	Logic-Controlled Clock Output
Symbol	Definition
<b>INSTRUCTION OPERAND SYMBOLS</b>	
d	4-bit Operand Field, 0-15 binary (RAM Digit Select)
r	2-bit Operand Field, 0-3 binary (RAM Register Select)
a	10-bit Operand Field, 0-1024 binary (ROM Address)
y	4-bit Operand Field, 0-15 binary (Immediate Data)
RAM(s)	Contents of RAM location addressed by s
ROM(t)	Contents of ROM location addressed by t
<b>OPERATIONAL SYMBOLS</b>	
+	Plus
-	Minus
→	Replaces
↔	Is exchanged with
=	Is equal to
A	The ones complement of A
⊕	Exclusive-OR
:	Range of values

**Table 6-4. Alphabetical Mnemonic Index of COP400 Instructions (continued)**

Instruction	Hexadecimal Opcode	Description
CQMA*	33/2C	Copy A to RAM, A
ING*	33/2A	INput G ports to A
INIL*	33/00	INput IL latches to A
ININ	33/28	INput IN inputs to A
INL*	33/2E	INput L ports to M, A
JID	FF	Jump INDirect
JMP*	60-63/00-FF	JuMP
JP	80-BE,CO-CE	Jump within Page
JSR*	68-6B/00-FF	Jump to SubRoutine
JSRP	80/BE	Jump to SubRoutine Page
LBI 0,9-15,0	08-0F	Load B Immediate (single-byte)
LBI 1,9-15,0	18-1F	
LBI 2,9-15,0	28-2F	
LBI 3,9-15,0	38-3F	Load B Immediate (double-byte)
LBI* 0,1-8	33/81-88	
LBI* 1,1-8	33/91-98	
LBI* 2,1-8	33/A1-A8	Load B Immediate (double-byte)
LBI* 3,1-8	33/B1-B8	
LD 0,1,2,3	05,15,25,35	Load RAM into A
LDD* 0,3,0-15	23/00-3F	LoaD A with RAM, Directly
LEI* 0-15	33/60-6F	Load EN Immediate
LQID	BF	Load Q INDirect
NOP	44	No OPERATION
OBD*	33/3E	Output Bd to D outputs
OGI*	33/50-5F	Output to G ports Immediate
OMG*	33/3A	Output RAM to G ports
RC	32	Reset C
RET	48	RETurn
RETSK	49	RETurn then SKip
RMB 0,1,2,3	4C,45,42,43	Reset Memory Bit
SC	22	Set C
SMB 0,1,2,3	4D,47,46,48	Set Memory Bit
SKC	20	SKip if C is true
SKE	21	SKip if A Equals RAM
SKGBZ* 0,1,2,3	33/01,11,03,13	SKip if G Bit is Zero
SKGZ*	33/21	SKip if G equals Zero (all 4 bits)
SKMBZ 0,1,2,3	01,11,03,13	SKip if Memory Bit is Zero
SKT	41	SKip on Timer
STII	70-7F	STore memory Immediate and Increment Bd
X 0,1,2,3	06,16,26,36	eXchange RAM with A
XABR	12	eXchange A with Br
XAD* 0,3,0-15	23/80-BF	eXchange A with RAM Directly
XDS 0,1,2,3	07,17,27,37	eXchange RAM with A and Decrement Bd
XIS 0,1,2,3	04,14,24,34	eXchange RAM with A and Increment Bd
XOR	02	eXclusive-OR A with RAM

**Table 6-4. Alphabetical Mnemonic Index of COP400 Instructions**

Instruction	Hexadecimal Opcode	Description
ADD	31	ADD A to RAM
ADT	4A	ADD Ten to A
AISC 1-15	51-5F	Add Immediate, Skip on Carry
ASC	30	Add with carry, Skip on Carry
CAB	50	Copy A to Bd
CAMQ*	33/3C	Copy A, RAM to Bd
CASC	10	Complement and Add with carry, Skip on Carry
CBA	4E	Copy Bd to A
CLRA	00	CLear A
COMP	40	ones COMPLEMENT of A to A

\*Double-Byte Instruction: first byte/second byte (or first byte range/second byte range).  
 \*\*Instruction not available or has different features on COP421-series.

Table 6-5. Table of COP400 Instructions Listed by Opcodes (Hexadecimal)

00	CLRA	3A	LBI 3,11	6F	invalid for	5F	OGI 15
01	SKMBZ 0	3B	LBI 3,12		COP420, COP410	60	LEI 0
02	XOR	3C	LBI 3,13	70	STII 0	61	LEI 1
03	SKMBZ 2	3D	LBI 3,14	71	STII 1	62	LEI 2
04	XIS 0	3E	LBI 3,15	72	STII 2	63	LEI 3
05	LD 0	3F	LBI 3,0	73	STII 3	64	LEI 4
06	X 0	40	COMP	74	STII 4	65	LEI 5
07	XDS 0	41	SKT	75	STII 5	66	LEI 6
08	LBI 0,9	42	RMB 2	76	STII 6	67	LEI 7
09	LBI 0,10	43	RMB 3	77	STII 7	68	LEI 8
0A	LBI 0,11	44	NOP	78	STII 8	69	LEI 9
0B	LBI 0,12	45	RMB 1	79	STII 9	6A	LEI 10
0C	LBI 0,13	46	SMB 2	7A	STII 10	6B	LEI 11
0D	LBI 0,14	47	SMB 1	7B	STII 11	6C	LEI 12
0E	LBI 0,15	48	RET	7C	STII 12	6D	LEI 13
0F	LBI 0,0	49	RETSK	7D	STII 13	6E	LEI 14
10	CASC	4A	ADT	7E	STII 14	6F	LEI 15
11	SKMBZ 1	4B	SMB 3	7F	STII 15	81	LBI 0,1
12	XABR	4C	RMB 0	80-BE	JP to word XX (0-3F <sub>16</sub> ) or JSRP to page 2, word XX (0-3F <sub>16</sub> ): opcode = 80 + XX	82	LBI 0,2
13	SKMBZ 3	4D	SMB 0			83	LBI 0,3
14	XIS 0	4E	CBA			84	LBI 0,4
15	LD 1	4F	XAS			85	LBI 0,5
16	X 1	50	CAB			86	LBI 0,6
17	XDS 1	51	AISC 1	BF	LQID	87	LBI 0,7
18	LBI 1,9	52	AISC 2	C0-CE	JP to word XX (0-3F <sub>16</sub> ): opcode = C0 + XX	88	LBI 0,8
19	LBI 1,10	53	AISC 3			91	LBI 1,1
1A	LBI 1,11	54	AISC 4	FF	JID	92	LBI 1,2
1B	LBI 1,12	55	AISC 5			93	LBI 1,3
1C	LBI 1,13	56	AISC 6			94	LBI 1,4
1D	LBI 1,14	57	AISC 7			95	LBI 1,5
1E	LBI 1,15	58	AISC 8			96	LBI 1,6
1F	LBI 1,0	59	AISC 9			97	LBI 1,7
20	SKC	5A	AISC 10	01	SKGBZ 0	98	LBI 1,8
21	SKE	5B	AISC 11	03	SKGBZ 2	A1	LBI 2,1
22	SC	5C	AISC 12	11	SKGBZ 1	A2	LBI 2,2
23	LDD/XAD**	5D	AISC 13	13	SKGBZ 3	A3	LBI 2,3
24	XIS 2	5E	AISC 14	21	SKGZ	A4	LBI 2,4
25	LD 2	5F	AISC 15	28	ININ	A5	LBI 2,5
26	X 2	60	JMP*** to Page 0, 1, 2, or 3	29	INIL	A6	LBI 2,6
27	XDS 2			2A	ING	A7	LBI 2,7
28	LBI 2,9	61	JMP*** to Page 4, 5, 6, or 7	2C	CQMA	A8	LBI 2,8
29	LBI 2,10			2E	INL	A8	LBI 2,8
2A	LBI 2,11	62	JMP*** to Page 8, 9, 10, or 11	3A	OMG	B1	LBI 3,1
2B	LBI 2,12			3C	CAMQ	B2	LBI 3,2
2C	LBI 2,13	63	JMP*** to Page 12, 13, 14, or 15	3E	OBQ	B3	LBI 3,3
2D	LBI 2,14			50	OGI 0	B4	LBI 3,4
2E	LBI 2,15	64	invalid	51	OGI 1	B5	LBI 3,5
2F	LBI 2,0	65	invalid	52	OGI 2	B6	LBI 3,6
30	ASC	66	invalid	53	OGI 3	B7	LBI 3,7
31	ADD	67	invalid	54	OGI 4	B8	LBI 3,8
32	RC	68	JSR*** to Page 0, 1, 2, or 3	55	OGI 5		
33	TWO WORD* (except LDD, XAD, JMP, JSR)	69	JSR*** to Page 4, 5, 6, or 7	56	OGI 6	**LDD/XAD Instruction First Word: 23; Second Word:	
		6A	JSR*** to Page 8, 9, 10, or 11	57	OGI 7	**00	LDD 0,0
34	XIS 3			58	OGI 8	01	LDD 0,1
35	LD 3	6B	JSR*** to Page 12, 13, 14, or 15	59	OGI 9	02	LDD 0,2
36	X 3			5A	OGI 10	03	LDD 0,3
37	XDS 3	6C		5B	OGI 11	04	LDD 0,4
38	LBI 3,9	6D	invalid for COP420, COP410	5C	OGI 12	05	LDD 0,5
39	LBI 3,10	6E		5D	OGI 13	06	LDD 0,6
				5E	OGI 14		

Table 6-5. Table of COP400 Instructions Listed by Opcodes (Hexadecimal) (continued)

07	LDD 0,7	28	LDD 2,8	89	XAD 0,9	A5	XAD 2,5
08	LDD 0,8	29	LDD 2,9	8A	XAD 0,10	A6	XAD 2,6
09	LDD 0,9	2A	LDD 2,10	8B	XAD 0,11	A7	XAD 2,7
0A	LDD 0,10	2B	LDD 2,11	8C	XAD 0,12	A8	XAD 2,8
0B	LDD 0,11	2C	LDD 2,12	8D	XAD 0,13	A9	XAD 2,9
0C	LDD 0,12	2D	LDD 2,13	8E	XAD 0,14	AA	XAD 2,10
0D	LDD 0,13	2E	LDD 2,14	8F	XAD 0,15	AB	XAD 2,11
0E	LDD 0,14	2F	LDD 2,15	90	XAD 1,0	AC	XAD 2,12
0F	LDD 0,15	30	LDD 3,0	91	XAD 1,1	AD	XAD 2,13
10	LDD 1,0	31	LDD 3,1	92	XAD 1,2	AE	XAD 2,14
11	LDD 1,1	32	LDD 3,2	93	XAD 1,3	AF	XAD 2,15
12	LDD 1,2	33	LDD 3,3	94	XAD 1,4	B0	XAD 3,0
13	LDD 1,3	34	LDD 3,4	95	XAD 1,5	B1	XAD 3,1
14	LDD 1,4	35	LDD 3,5	96	XAD 1,6	B2	XAD 3,2
15	LDD 1,5	36	LDD 3,6	97	XAD 1,7	B3	XAD 3,3
16	LDD 1,6	37	LDD 3,7	98	XAD 1,8	B4	XAD 3,4
17	LDD 1,7	38	LDD 3,8	99	XAD 1,9	B5	XAD 3,5
18	LDD 1,8	39	LDD 3,9	9A	XAD 1,10	B6	XAD 3,6
19	LDD 1,9	3A	LDD 3,10	9B	XAD 1,11	B7	XAD 3,7
1A	LDD 1,10	3B	LDD 3,11	9C	XAD 1,12	B8	XAD 3,8
1B	LDD 1,11	3C	LDD 3,12	9D	XAD 1,13	B9	XAD 3,9
1C	LDD 1,12	3D	LDD 3,13	9E	XAD 1,14	BA	XAD 3,10
1D	LDD 1,13	3E	LDD 3,14	9F	XAD 1,15	BB	XAD 3,11
1E	LDD 1,14	3F	LDD 3,15	A0	XAD 2,0	BC	XAD 3,12
1F	LDD 1,15	80	XAD 0,0	A1	XAD 2,1	BD	XAD 3,13
20	LDD 2,0	81	XAD 0,1	A2	XAD 2,2	BE	XAD 3,14
21	LDD 2,1	82	XAD 0,2	A3	XAD 2,3	BF	XAD 3,15
22	LDD 2,2	83	XAD 0,3	A4	XAD 2,4		
23	LDD 2,3	84	XAD 0,4				
24	LDD 2,4	85	XAD 0,5				
25	LDD 2,5	86	XAD 0,6				
26	LDD 2,6	87	XAD 0,7				
27	LDD 2,7	88	XAD 0,8				

\*\*\*00 + XX JSR or JMP to page 0, 4, 10, or 14, word XX (03F<sub>16</sub>): 0-3F  
 40 + XX JSR or JMP to page 1, 5, 11, or 15, word XX (0-3F<sub>16</sub>): 40-7F  
 80 + XX JSR or JMP to page 2, 6, 12, or 16, word XX (0-3F<sub>16</sub>): 80-BF  
 C0 + XX JSR or JMP to page 3, 7, 13, or 17, word XX (0-3F<sub>16</sub>): C0-FF

**.SPACE DIRECTIVE**

Syntax: [<label>].SPACE<expression>  
 [<comments>]

The .SPACE directive skips forward a number of lines on the output listing as specified by the expression in the operand field.

Example:

```
.SPACE 20
Skip 20 lines.
```

**.FORM DIRECTIVE**

Syntax: .FORM ['<string>'] [<comments>]

The .FORM directive spaces forward to the top of the next page of the output listing (form feed). The optional string is printed as a page title on each page until a .FORM directive containing a new string is encountered. No action is taken (except for a new page title) if the .FORM directive is encountered immediately after an assembler-generated top-of-page request which occurs when an output listing is full.

Example:

```
.FORM 'BCD ARITHMETIC ROUTINES'
;FORM FEED
```

The string must be 26 or fewer characters to be fully printed on the output listing.

**.WORD DIRECTIVE**

Syntax: [<label>:].WORD<expression>  
 [<expression>]. . . [<comments>]

The .WORD directive stores consecutively in memory one 8-bit byte of data for each given expression. If the directive has a label, it refers to the address of the first expression. The value of each expression must be in the range -128 to +127 for signed data or 0 to 255 for unsigned data.

The hexadecimal value of ASCII characters may be stored in memory using the .WORD directive and an operand expression specifying character strings or their hexadecimal equivalents. (See Table 6-8, ASCII Character Set in Hexadecimal Representation.)

In the smaller system dedicated applications in which COP400 devices are commonly used, a more typical function of the .WORD directive is to place 7-segment decode data in ROM for output to the digits of an LED or VF display. Table 6-9 provides the 7-segment binary and hexadecimal values associated with the display numerals 0 through 9, with and without the Decimal

Point bit on and with the contents of ROM (L<sub>7</sub>-L<sub>0</sub>) assigned to Sa-Sg, D.P. as well as to D.P., Sg-Sa.

Examples:

1. .WORD X'FF
2. TBL: .WORD MPR-10, X'FF
3. .WORD 'H', 'E', 'L', 'O'
4. .WORD X'48, X'45, X'4C, X'4C, 2'4F

Example 1 stores the hexadecimal number FF in a byte of memory.

Example 2 stores two hexadecimal numbers in consecutive bytes in memory.

Examples 3 and 4 store the hexadecimal value of the word HELLO in consecutive bytes of memory.

**.ADDR DIRECTIVE**

Syntax: [<label:>].ADDR<expression>  
[,<expression>]... [<comments>]

The .ADDR directive generates 8-bit bytes as specified by one or more expressions in the operand field of this directive and places them in successive memory locations. These expressions are usually labels and are used as address pointers by the COP400 JID (Jump Indirect) instruction which transfers program control to the contents of the address generated by the .ADDR directive.

**Table 6-6. Summary of Assembler Directives**

Directive	Function	Section
.ADDR	Address constant generation	6.4.3
.CHIP	Identification of COP400 device	6.4.3
.DO	Begin Macro-time looping*	6.5.7
.ELSE	Conditional assembly directive	6.5.5
.END	Physical end of source program	6.4.3
.ENDDO	End Macro-time looping*	6.5.7
.ENDIF	Conditional assembly directive	6.5.5
.ENDM	End Macro definition*	6.5.1
.ERROR	Macro error message generation**	6.5.6
.EXIT	Exit DO loop*	6.5.7
.FORM	Output listing top-of-form	6.4.3
.IF	Conditional assembly directive	6.5.5
.IFC	Macro conditional assembly**	6.5.5
.INCLD	Include disk file source code	6.4.3
.LIST	Listing output control	6.4.3
.LOCAL	Establish a new local symbol region	6.4.3
.MACRO	Begin Macro definition*	6.5.1
.MDEL	Macro delete**	6.5.6
.MLOC	Macro local symbol designation*	6.5.5
.OPT	Define COP400 device options	6.4.3
.PAGE	Set assembler location counter to page address	6.4.3
.SET	Assign values to variables	6.4.3, 6.5.6
.SPACE	Space n lines on Output Listing	6.4.3
.TITLE	Identification of program	6.4.3
.WORD	8-bit data generation	6.4.3

\*Used only in Macro definitions.

\*\*Macro related directives.

**Table 6-7. List Options**

Control Function	Bit		6-Bit Hex Value	Description
	Positions	Binary Value		
Master List	0	0	00	Suppress all listing
		1	01	*Full listing
.IF List	1	0	00	*Suppress listing of unassembled code
		1	02	Full listing (of .IFs and IFCs)
Macro List	2,3	00	00	*List only macro calls
		10	08	List only code generated by macro calls
		11	0C	List all code expanded during macro calls
Binary List	4	0	00	List only the first two bytes of generated data
		1	10	*List all the binary output by statements generating more than one word (e.g., ASCII)
Include List	5	0	00	*List only error lines for the included file
		1	20	List the included file (source statements from the included files are listed without line numbers)

\*Indicates Default

Table 6-8. ASCII Character Set in Hexadecimal Representation

Character	7-Bit Hex Number	Character	7-Bit Hex Number	Character	7-Bit Hex Number	Character	7-Bit Hex Number
NUM	00	SP	20	@	40		60
SOH	01	!	21	A	41	a	61
STX	02	"	22	B	42	b	62
ETX	03	#	23	C	43	c	63
EOT	04	\$	24	D	44	d	64
ENQ	05	%	25	E	45	e	65
ACK	06	&	26	F	46	f	66
BEL	07	'	27	G	47	g	67
BS	08	(	28	H	48	h	68
HT	09	)	29	I	49	i	69
LF	0A	*	2A	J	4A	j	6A
VT	0B	+	2B	K	4B	k	6B
FF	0C	,	2C	L	4C	l	6C
CR	0D	-	2D	M	4D	m	6D
SO	0E	.	2E	N	4E	n	6E
SI	0F	/	2F	O	4F	o	6F
DLE	10	0	30	P	50	p	70
DC1	11	1	31	Q	51	q	71
DC2	12	2	32	R	52	r	72
DC3	13	3	33	S	53	s	73
DC4	14	4	34	T	54	t	74
NAK	15	5	35	U	55	u	75
SYN	16	6	36	V	56	v	76
ETB	17	7	37	W	57	w	77
CAN	18	8	38	X	58	x	78
EM	19	9	39	Y	59	y	79
SUB	1A	:	3A	Z	5A	z	7A
ESC	1B	;	3B	[	5B		7B
FS	1C	<	3C	\	5C		7C
GS	1D	=	3D	]	5D	ALT	7D
RS	1E	>	3E	↑	5E	ESC	7E
US	1F	?	3F	←	5F	DEL, rubout	7F

This directive masks out the upper eight bits of the expression specified in the operand field, and places the lower eight bits in successive memory locations. Next, the lower eight bits of the symbol or expression are masked and a comparison is made of the upper eight bits with the current location counter address to ensure that the address generated by the .ADDR directive is in the same 4-page ROM block as the assembler location counter — this test is necessary since the JID instruction must access a pointer and transfer program control within the current 4-page program ROM block. If this test indicates an out-of-range expression, an error message will be generated upon assembly and listed on the assembler listing. For further information on the operation, restrictions associated with, and use of the COP400 JID instruction, see the MOS Data Book or the specific Data Sheet.

**Example:**

Create an address pointer table to be used by the COP400 JID instruction.

Assuming that program labels TBL1, TBL2 and TBL3 are located at memory locations 01D3, 01DF and 02C0, respectively, with the .ADDR directive placed

in the program source code preceding memory location 01C0 using an Assignment Statement, then

```
. = X'1C0 ;SET LOCATION POINTER TO
;ROM LOCATION X'01C0
```

```
.ADDR TBL1,TBL2, TBL3
```

will place the following address pointer data in the following memory locations:

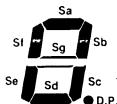
Address (HEX)	Data (HEX)	
01C0	D3	(lower eight bits of address of TBL1 label)
01C1	DF	(lower eight bits of address of TBL2 label)
01C2	XX	(ERROR message will be generated — TBL3 address is out of range for .ADDR directive)

**.PAGE DIRIRECTIVE**

Syntax: .PAGE [<expression>][;<comments>]

The .PAGE directive changes the assembler's location counter to the address of the beginning of the ROM page specified by the expression in the operand field.

**Table 6-9. Display Digit Segments**



Binary Values								Hexadecimal Values		Hexadecimal Values	
Sa-Sg, D.P. → L <sub>7</sub> -L <sub>0</sub>								Sa-Sg, D.P. → L <sub>7</sub> -L <sub>0</sub>		D.P., Sg-Sa → L <sub>7</sub> -L <sub>0</sub>	
Sa	Sb	Sc	Sd	Se	Sf	Sg	D.P. Off/On	Off	On	Off	D.P. On
1	1	1	1	1	1	0	0/1	FC	FD	3F	BF
0	1	1	0	0	0	0	0/1	60	61	06	86
1	1	0	1	1	0	1	0/1	DA	DB	5B	DB
1	1	1	1	0	0	1	0/1	F2	F3	4F	CF
1	1	1	0	0	1	1	0/1	66	67	66	E6
1	0	1	1	0	1	1	0/1	B6	B7	6D	ED
1	0	1	1	1	1	1	0/1	B7	BF	7D	FD
1	1	1	0	0	0	0	0/1	E0	E1	07	87
1	1	1	1	1	1	1	0/1	FE	FF	7F	FF
1	1	1	0	0	1	1	0/1	E6	E7	67	E7





The value of the expression may not exceed the maximum ROM page number for the chip being used. (See .CHIP directive.) There are 64 locations in each ROM page.

Example:

```
.PAGE 2          ;SET LOCATION COUNTER TO
                  ;X'80
```

**.LOCAL DIRECTIVE**

Syntax: .LOCAL [<comments>]

The .LOCAL directive establishes a new program section for local labels (labels beginning with a dollar sign [\$]). All local labels between two .LOCAL directive statements have their values assigned to them only within that particular section of the program. Note that a .LOCAL directive is assumed at the beginning and the end of a program; thus, one .LOCAL directive within a program divides the program into two local sections. Up to 58 .LOCAL directives may appear in one assembly.

Example:

```
$X:.WORD 1      ;FIRST LABEL $X
.LOCAL          ;ESTABLISH NEW LOCAL
                ;SYMBOL SECTION
$X:.WORD 1      ;SECTION LABEL $X, NO CON-
                ;FUSION SINCE THEY ARE IN
                ;DIFFERENT "LOCAL" BLOCKS
```

**.SET DIRECTIVE**

Syntax: .SET <symbol>,<expression> [<comments>]

The .SET directive is used to assign values to symbols. In contrast to an ASSIGNMENT statement, a symbol assigned a value with the .SET directive can be assigned different values an arbitrary number of times within an assembly language program with each new value taking precedence over the previous value for a particular symbol.

Example:

```
.SET A,100      ;SET A = 100
.SET B,50       ;SET B = 50
.SET C,A-25*B/4 ;SET C = A-25*B/4
```

Note: this expression is always evaluated from left to right regardless of the operators used between the variables and constants unless parentheses appear in the expression.

**.CHIP DIRECTIVE**

Syntax: .CHIP<expression>[<comments>]

The .CHIP directive specifies to the assembler the particular COP device for which the assembly source code is being written. This is necessary since different COP400 devices having a different number of COP400 instructions may use the COP Cross Assembler. The devices which may be specified with the .CHIP directive and the corresponding values for their operand field expressions are as follows:

COP400 Device	Operand Expression
COP410P	410
COP411L	411
COP420/420L/420C	420*
COP421/421L/421C	421
COP422/422L	422
COP440	440
COP441	441
COP442	442
COP444L	444
COP445L	445
COP2440	2440
COP2441	2441
COP2442	2442

\*Indicates default value.

A feature associated with the .CHIP directive is that the assembler allows for multiple .CHIP directives in the program. The assembler will treat the program as one written for the COP device specified by the last .CHIP directive (the default device is the COP420) until it encounters a new .CHIP directive. It will then treat the program as one written for a different device as specified by the new .CHIP directive.

Example:

1. No .CHIP directive:

```
.PAGE 0
.
.END
```

Assembler assumes default device, the COP420.

2. Multiply .CHIP directives:

```
.PAGE 0 ;ASSEMBLER ASSUMES COP420
.
.CHIP 440 ;ASSEMBLER ASSUMES COP440
. ;FOR FOLLOWING CODE UNTIL
. ;NEXT .CHIP
.END
```

**.OPT DIRECTIVE**

Syntax: .OPT<expression<sub>1</sub>>,<expression<sub>2</sub>> [<comments>]

The .OPT directive specifies to the assembler which mask-programmable options have been selected for the device for which the program is written (as specified by the .CHIP directive). The first expression indicates the option number; the second expression indicates the value to be assigned to the specified option number. Values for the first expression (option numbers) must be within the range 1 through 56; values for the second expression (option values) must be within the range 0 through 14. A value of 15 indi-

icates an undefined option. Also, option numbers and values must be valid for the particular COP device for which the program is written. For specific information on the options and values associated with COP400 devices, see the MOS Data Book or the specific Data Sheet.

The .OPT directive does not convey information to the assembler for its own use. It is necessary to provide option information to be included in the assembler Load Module output file for mask-programming the selected options into the COP part when fabricated.

Example:

```
.OPT 1,3      ;SPECIFY OPTION 1 = 3
.OPT 2,1      ;SPECIFY OPTION 2 = 1
```

#### .INCLD DIRECTIVE

Syntax: .INCLD<filename>[;<comments>]

The .INCLD directive includes the symbolic file specified in the operand field of the directive in the current assembler source code. Specifically, it causes the assembler to read source code from the specified file on the current diskette until an end-of-file mark is reached, at which time it will again start reading source code from the assembly input file. The file must be a symbolic file. The default modifier is SRC. Since the specified file is included in the source code at assembly time, the included file must, as mentioned above, be contained on the current diskette as assembly time. Expansion of the source code included by this directive on the assembler output listing is controlled by bit 5 in the operand field of the .LIST directive. A .LIST with bit 5 set to "1" must be contained in the assembly source code prior to the .INCLD directive in order for the contents of the included file to be expanded on the assembler output listing (see .LIST directive, above).

Example:

```
.LIST X'21      ;EXPANDING .INCLD SOURCE
                ;CODE ON OUTPUT LISTING
.INCLD BCDADD  ;INCLUDE 'BCDADD.SRC' FILE
                ;ON CURRENT DISKETTE
```

#### CONDITIONAL ASSEMBLY DIRECTIVES

Syntax: [<label>:].IF<expression>[;<comments>]  
           .ELSE          [;<comments>]  
           .ENDIF       [;<comments>]

The conditional assembly directives selectively assemble portions of a source program based on the value of the expression in the operand field of the .IF directive statement. All source statements between an .IF directive and its associated .ENDIF are defined as an .IF-.ENDIF block. These blocks may be nested to a depth of ten. The .ELSE directive can be optionally included in an .IF-.ENDIF block. The .ELSE directive divides the block into two parts. The first part of the source statements block is assembled if the .IF expression is greater than zero; otherwise, the second part is assembled. When the .ELSE directive is not included in a block, the block is assembled only if the .IF expression is greater than zero. If an error is detected in the expression, the assembler assumes a true value (greater than zero).

Example:

1. Two part conditional assembly:

```
.IF COMPR
. . .
.ELSE
. . .
.ENDIF
```

Assembled if COMPR greater than zero

Assembled if COMPR less than or equal to zero

2. Nested .IF-.ENDIF block conditional assembly:

```
.IF SMT
. . .
.ELSE
. . .
.ELSE
. . .
.ENDIF
.ELSE
. . .
.ENDIF
```

Assembled if SMT greater than zero

Assembled if SMT less than or equal to zero

Assembled if OBR is greater than zero and SMT is less than or equal to zero

Labels appearing on .IF statements are assigned the address of the next assembled instructions. Labels cannot be used on .ELSE or .ENDIF directives.

Listing of conditional assembly code is controlled by the .LIST directive.

#### 6.5 Macros

The primary use of macros is to make the assembly process easier, by inserting duplicative or similar assembly language statements into the program source code without the need to manually enter these statements into the program each time they are required. A macro, once defined, will automatically, during assembly time, place reiterative code or similar code with changed parameters into the assembler source code when called by its macro name. The following sections are devoted to explaining the process of defining and calling macros, with and without parameters, and describing assembler directives associated with the use of macros.

Using macros, a programmer can gradually build a library of basic routines, allowing variables unique to particular programming applications to be defined in and passed to a particular macro when called by main programs. Such macros can be automatically included in the assembly source code of main programs using the .INCLD directive (see Section 5.4.3) or read into the source code during an editing session using the READ FROM<filename> command (see Chapter 5).

### 6.5.1 Defining a Macro

The process of defining a macro involves preparing statements which perform the following functions:

- Give it a name
- Declare any parameters to be used
- Write the assembler statements it contains
- Establish its boundaries

Macros must be defined before their use in a program. Macro definitions within an assembly do not generate code. Code is generated only when macros are called by the main program. Macro definitions are formed as follows:

```
.MACRO  mname [,parameters]
.
.
macro body
.
.
.ENDM
```

where,

- a. .MACRO is the directive mnemonic which initiates the macro definition. It must be terminated by at least one blank.
- b. "mname" is the name of the macro. It is legal to define a macro with the same name as an already existing macro, in which case the latest definition is operative. Previous definitions are, however, retained in the macro definition table unless deleted from the buffer space by the .MDEL directive (see below). The macro name is used by the main program to call the macro, and must adhere to the rules given for symbol construction in Section 6.2.
- c. [,parameters] is the optional list of parameters used in the macro definition. Each parameter must adhere to the symbol construction rules. Parameters are delimited from "mname" and successive parameters by commas.
- d. The macro body consists of assembly language statements. The macro body may consist of simple text, text with parameters, and/or macro-time operators.
- e. The .ENDM signifies the end of the macro and must be used to terminate a macro definition.

The following are examples of legal and illegal .MACRO directives.

Legal	Illegal	Reason Illegal
.MACRO MAC,A,B	.MACRO SUB,?1H	Special character used in parameter
.MACRO \$ADD,OP1,OP2	.MACRO 1MAC,C,D	First character of macro name numeric
.MACRO LIST,\$1	.MACRO MAC,25	First character of parameter must be alphabetic
.MACRO MSG3	.MACRO M\$AC	Special character used in macro name

### SIMPLE MACROS

The simplest form of macro definition is one with no parameters or macro operators. The macro body is simply a sequence of assembly language statements which are substituted for each macro call. Of course, such identical macro calls are inefficient if called repetitively within the same assembly program — a repeatedly used series of assembly language statements within a program should be coded as a subroutine. However, simple macros with no variables are useful in compiling a library of basic routines to be used within different programs, since, as mentioned above, they allow the programmer to simply call the macro within the program rather than repeatedly coding all the macro body statements into each program when needed. An example of a simple macro definition follows:

```
;MACRO "INC2" TO INCREMENT A 2-DIGIT BCD
;RAM COUNTER WHEN CALLED, B MUST POINT TO
;A LOW-ORDER DIGIT OF COUNTER
.MACRO  INC2 ;BEGIN MACRO DEFINITION
SC          ;INITIALIZE C TO 1 TO ADD
            ;LOW-ORDER DIGIT
CLRA        ;ZERO TO A
AISC 6      ;BCD ADJUST RESULT IF
            ;NECESSARY
ASC
ADT          ;IF RESULT > 9, LOW-ORDER
            ;DIGIT = 0
XIS          ;PLACE INCREMENTED DIGIT IN
            ;M, POINT TO HIGH-ORDER DIGIT
CLRA        ;ZERO TO A
AISC 6      ;ADD CARRY, IF PROPAGATED
            ;FROM LOW-ORDER DIGIT TO
            ;HIGH-ORDER DIGIT
ASC
ADT          ;BCD ADJUST RESULT IF
            ;NECESSARY
X            ;REPLACE DIGIT IN M
.ENDM
```

### MACROS WITH PARAMETERS

Obviously, the above macro could be made more flexible by the addition of parameters in the macro definition, allowing the programmer to specify the low-order digit of the RAM counter to be incremented in the macro call itself, rather than relying on the instruction in the main program which loads the B (RAM address) register with the proper value before calling the macro. The following is an example of the use of parameters within a macro definition to accomplish this result:

```

;MACRO "INC2A" TO INCREMENT A 2-DIGIT RAM
;COUNTER THE LOW-ORDER DIGIT OF THE
;COUNTER IS REPRESENTED BY PARAMETERS
;"R", "D"
.MACRO INC2A, ;R,D = REGISTER #. DIGIT # OF
R,D ;LOW-ORDER DIGIT COUNTER
LBI ;POINT TO LOW-ORDER DIGIT
;OF COUNTER
SC ;INITIALIZE C TO A TO ADD TO
;LOW-ORDER DIGIT
CLRA ;ZERO TO A
AISC 6 ;BCD ADJUST RESULT IF
;NECESSARY
ASC
ADT ;IF RESULT >9, LOW-ORDER
;DIGIT = 0
XIS ;PLACE INCREMENTED DIGIT IN
;M, POINT TO HIGH-ORDER DIGIT
CLRA ;ZERO TO A
AISC 6 ;ADD CARRY, IF PROPAGATED
;FROM LOW-ORDER DIGIT TO
;HIGH-ORDER DIGIT
ASC
ADT ;BCD ADJUST RESULT IF
;NECESSARY
X ;REPLACE DIGIT IN M
.ENDM ;END MACRO DEFINITION

```

### 6.5.2 Calling a Macro

Once a macro has been defined, it may be called by a program to generate code. A macro is called by placing the macro name in the operand field of the assembly language statement and the actual value or parameters to be used (if any) by the symbolic macro definition parameters. The following form is used for a macro call:

<mname>[<parameters>]

where,

- a. "mname" is the name previously assigned in the macro definition.
- b. [<parameters>] is the list of input parameters. When a macro is defined without parameters, the parameter list is omitted from the call.

A call to the simple INC2 macro, defined above, would be expanded as follows:

Source Program Before Assembly		Assembled Program (shown without comments)
		.
		.
		INC2
		SC
		CLRA
		AISC 6
INC2	generates	ADT
		XIS
		CLRA
		AISC 6
		ASC
		ADT
		X
		.
		.

Note: The macro call (INC2), as well as the expanded macro machine and source code will appear on the assembler output listing if a .LIST directive with bits 2 and 3 set is placed in the program's source code (see Section 6.4.3). The macro call statement (INC2) itself will not generate machine code.

### 6.5.3 Using Parameters

As already indicated, the power of a macro can be increased tremendously through the use of optional parameters. The parameters allow variable values to be declared when the macro is called.

For example, the parameter version of INC2, INC2A (Section 6.5.1), could be used to increment a 2-digit RAM based upon the parameter values specified in the macro call. The following macro call illustrates the use of the INC2A macro to increment a 2-digit RAM counter whose low-order digit is contained in RAM register 3, digit 14.

Source Program Before Assembly		Assembled Program (shown without comments)
		.
		.
		INC2A 3,14
		LBI 3,14
		SC
		CLRA
		AISC 6
		ASC
INC2A 3,14	generates	ADT
		XIS
		CLRA
		AISC 6
		ASC
		ADT
		X
		.
		.

When parameters are included in a macro call, the following rules apply to the parameter list:

- a. Commas or blanks delimit parameters.
- b. Consecutive blanks are treated as a single delimiter.
- c. A leading, following or embedded comma in a string of blanks is treated as a single delimiter.
- d. A semicolon terminates the parameter list and starts the comment field.
- e. Quotes may be included as part of a parameter except as the first character of a parameter.
- f. A parameter may be enclosed in single quotes ('), in which case the quotes are removed and the string is used as the parameter. This function is useful when blanks, commas, or semicolons are to be included in the parameter.
- g. To include a quote in a quoted parameter, it must be preceded by another quote ("").
- h. Missing or null parameters are treated as strings of length zero.

#### PARAMETERS REFERENCED BY NUMBER

"#" is a macro operator that references the parameter list in the macro call. When used in an expression, it is replaced by the number of parameters in the macro call. The following .IF directive, for example, causes the conditional code to be expanded if there are more than 10 parameters in the macro call:

```
.IF #>10
    '#N' — Nth Parameter
```

When used in conjunction with a constant or variable, the '#' operator references individual parameters in the parameter list. The following example demonstrates how this function may be used in defining and calling a macro to establish a program memory data table:

```
.MACRO X          ;MACRO DEFINITION
.WORD #1,#2,#3
.ENDM
Macro Call      Generated Code
X X'61,X'FF,X'90 .WORD X'61,X'FF,X'90
```

This technique eliminates the need for naming each parameter in the macro definition, particularly convenient when long parameter lists are to be used. It also allows powerful macros to be defined using an arbitrary number of parameters.

#### 6.5.4 'A' — Concatenation Operator

The "A" macro operator is used for concatenation. When found, the "A" is removed from the output string and the strings on each side of the operator are compressed together after parameter substitution.

Example:

```
.MACRO LABEL,X
R X: .WORD 1
I I: .WORD 1
```

The Macro call: .

```
LABEL 0
```

generates: .

```
RO: .WORD 1
IO: .WORD 1
```

Another example of the use of this operation is shown in Section 6.5.8 (Macro-Time Loop Example).

#### 6.5.5 Local Symbols

```
Syntax: .MLOC <symbol>[,<symbol>]...
        [<comments>]
```

When a label is defined within a macro, a duplicate definition results with the second and each subsequent call of the macro. This problem can be avoided by using the .MLOC directive to declare labels local to the macro definition. In other words, if a macro definition containing fixed labels is to be called more than once during an assembly, duplicate definition errors will occur unless the .MLOC directive is used in the macro definition.

To illustrate this problem, consider the following macro definition, intended for multiple calls in an assembly, which does not use the .MLOC directive. Since it is a multiple loop routine, jumping back to the CLRA instruction, the inability to use a fixed label referencing this instruction requires the use of more complicated transfer of control instructions (JP) referenced to the assembly location counter (".") and not to one label:

```
;MACRO "CLRAM" TO CLEAR ALL DIGITS (0-15) OF
;ALL COP420 DATA MEMORY REGISTERS (0-3)
.MACRO CLRAM
LBI 3,0          ;CLEAR REGISTER 3 FIRST
CLRA
XIS              ;EXCHANGE ZEROS INTO
                ;MEMORY DIGIT
JP -2           ;JUMP BACK TO "CLRA" UNTIL
                ;REGISTER CLEARED
XABR            ;REGISTER CLEARED, BR TO A
AISC 15        ;REGISTER 0 CLEARED?
JP .+3         ;YES, JUMP TO FIRST INSTRU-
                ;TION AFTER ROUTINE
XABR            ;NO, BR -1 TO BR
JP .-7         ;JUMP BACK TO "CLRA" TO
                ;CLEAR NEXT REGISTER
.ENDM          ;END MACRO DEFINITION
```

Now here is the same macro (without comments) using the .MLOC directive which allows the fixed label, "CLEAR," to be referenced by the JP instructions:

```
.MACRO      CLRAM
.MLOC      CLEAR
           LBI 3,0

CLEAR:     CLRA
           XIS
           JP CLEAR
           XABR
           AISC 15
           JP .+3
           XABR
           JP CLEAR
           .ENDM
```

The .MLOC directive may occur at any point in a macro definition, but it must precede the first occurrence of the symbol(s) it declares local. If it does not, no error will be reported per se, but symbols used before the .MLOC will not be recognized as local. Local macro labels appear in the symbol table map at the end of the assembly listing as ZZXXXX, where XXXX is a particular hex number.

### 6.5.6 Conditional Expansion

The versatility and power of the macro assembler is enhanced by the conditional assembly directives. The conditional assembly directives (.IF, .ELSE, .ENDIF) allow the user to generate different lines of code from the same macro simply by varying the parameter values used in the macro calls. Three relational operators are provided:

= (equal) < (less than) > (greater than)

### .IF, .ELSE, .ENDIF DIRECTIVES

When the macro assembler encounters an .IF directive within a macro expansion, it evaluates the relational operation that follows. If the expression is satisfied (evaluated greater than 0), the lines following the .IF are expanded until an .ELSE or an .ENDIF is encountered. If the expression is not satisfied (evaluated less than or equal to 0), only the lines from the .ELSE to the .ENDIF are expanded. See Section 6.4.3 for additional information on the conditional assembly directives.

Example:

```
;SHIFT THE CONTENTS OF RAM ADDRESS R,D
;RIGHT IF N>0, LEFT OTHERWISE

.MACRO SHIFT R,D,N
LBI R,D      ;POINT TO RAM DIGIT R,D
.IF N>0
CLRA        ;SHIFT RIGHT IF N>0
SKMBZ 3
AISC 4
SKMBZ 2
AISC 2
SKMBZ 1
AISC 1
.ELSE      ;SHIFT LEFT IF N≤0
LD
ADD
.ENDIF
```

```
X          ;EXCHANGE SHIFTED DIGIT IN
          ;A BACK INTO RAM
.ENDM      ;END MACRO DEFINITION
```

### .IFC DIRECTIVE

Syntax: [<label>:].IFC<string<sub>1</sub>><operator>  
<string<sub>2</sub>>[;<comments>]

The .IFC directive allows conditional assembly based on character strings rather than the value of an expression as in the .IF directive. String<sub>1</sub> and string<sub>2</sub> are the character strings to be compared. Operator is the relational operator between the strings. Two operators are allowed: EQ (equal) and NE (not equal). If the relational operator is satisfied, the lines following the .IFC are assembled until an .ELSE or an .ENDIF is encountered. .ELSE and .ENDIF directives have the same effect with the .IFC directive as they do with the .IF directive.

The primary application of the .IFC is to compare a parameter value such as #3 against a specific string.

Example:

```
.IF #3 NE INTEGER
```

### 6.5.7 Useful Directives

Syntax: [<label>:].SET<symbol>,<expression>  
[;<comments>]

The .SET directive is used to assign values to symbols (variables). A variable assigned a value with the .SET directive can be reassigned different values an arbitrary number of times (see Section 6.4.3). Set variables are useful during macro expansion to control macro-time looping and macro communication. To ensure value correspondence between pass 1 and pass 2 of the assembler, all values in the expression must be defined before use in a .SET directive. If a value is not previously defined, an error is reported and a value of zero is returned. For an example of the .SET directive in a macro-time loop, see Section 6.5.8.

### .MDEL DIRECTIVE

Syntax: [<label>:].MDEL<mname>[,<mname>]...  
[;<comments>]

The .MDEL directive deletes macro definitions from the macro definition table and frees the buffer space used by the definitions.

Examples:

```
.MDEL INC2
```

### .ERROR DIRECTIVE

Syntax: [<label>:].ERROR [<string>]  
[;<comments>]

The .ERROR directive generates an error message and an assembly error that is included in the error count at the end of the program. The directive is useful for parameter checking in macros. For example, the INC2A macro, defined in Section 6.5.1, will put out erroneous code, if written for a COP420 program, if R>3 or D>15, since the COP420 has four RAM registers (0-3) containing 16 digits (0-15) each. To flag this condition with an error message, the following .ERROR directives may be included in the INC2A macro definition:

```

.MACRO INC2A,R,D
.IF D>15
.ERROR 'LBI WILL NOT WORK WITH D VALUE>15'
.ELSE
.IF R>3
.ERROR 'LBI WILL NOT WORK WITH R VALUE>3'
.ELSE
LBI R,D
SC
CLRA
AISC 6
ASC
ADT
XIS
CLRA
AISC 6
ASC
ADT
X
.ENDIF
.ENDIF
.ENDM

```

### 6.5.8 Macro-Time Looping

#### .DO AND .ENDDO DIRECTIVES

Syntax: [<label>:].DO<count>[;<comments>]  
 [<label>:].ENDDO[;<comments>]

Macro-time looping is facilitated through the .DO and .ENDDO directives. These directives are used to delimit a block of statements which are repeatedly assembled. The number of times the block will be assembled is specified by the .DO directive "count" value. Following is the format of a .DO-.ENDDO block:

```

.DO    count
      .
      .
      source
      .
      .
      .ENDDO

```

Note: .DO, .ENDDO, and .EXIT are defined only within a macro definition.

The "X" macro described in the section on "#" could be modified to generate a variable number of words, using .DO and a loop counter.

#### .EXIT DIRECTIVE

Syntax: [<label>:].EXIT [<comments>]

Early termination of looping in a .DO-.ENDDO block can be effected with the .EXIT directive. This directive allows the current loop to finish and then terminates looping. The .EXIT directive is commonly used in conjunction with a conditional test within a macro loop which will exit from the loop if a variable is equal to a particular value. In such cases the .DO "count" value is not crucial, provided it exceeds the maximum number of times the .DO loop will be required or expected to be

executed for a particular macro definition or for possible macro calls.

#### EXAMPLE OF A MACRO-TIME LOOP

The following examples show the use of the .DO, .ENDDO, and .EXIT directives. The macro CTAB generates a constant table from 0 to MAX where MAX is a parameter of the macro call. Each word has label DOX:, where X is the value of the data word.

```

.MACRO CTAB,MAX
.SET    X,0
.DO    MAX+1
DO X:  .WORD  X
      .SET    X,X+1
      .ENDDO
      .ENDM

```

Now a call of the form:

```
.CTAB 10
```

generates code equivalent to:

```

.SET    X,0
D00:  .WORD  X
      .SET    X,X+1
D01:  .WORD  X
      .SET    X,X+1
D02:  .WORD  X
      .
      .
      .
      .SET    X,X+1
D09:  .WORD  X
      .SET    X,X+1
D10:  .WORD  X

```

Note: Care must be taken when writing macros that generate a variable number of data words through the use of the .IF or the .DO directives. If the operands on these directives are forward referenced, their values change between pass 1 and pass 2 and the number of generated words may change. Should this be the case, all labels defined after the macro call that has changed values generate numerous assembly errors of the following form:

```
ERROR DUP .DEF
```

### 6.5.9 Nested Macro Calls

Nested macro calls are allowed; that is, a macro definition may contain a call to another macro. When a macro call is encountered during macro expansion, the state of the macro currently being expanded is saved and expansion begins on the nested macro. Upon completing expansion of the nested macro, expansion of the original macro continues. Depth of nesting allowed will depend on the parameters list sizes, but on the average about 10 levels of nesting will be allowed.





## MONITOR PROGRAM LISTING

```

1          .TITLE DSPLY, 'COP420 DISPLAY DEMO'
2 ;COP 420 DISPL 4Y/KEYBOARD DEBOUNCE/DECODE ROUTINE.
3 ;DISPLAYS 14 BCD DIGITS CONTAINED IN M(0,14) THROUGH
4 ;M(0,1), HIGH-ORDER TO LOW-ORDER, RESPECTIVELY.
5 ;DECIMAL POINT POSITION VALUE CONTAINED IN M(0,15).
6 ;DIGIT POSITION VALUE CONTAINED IN M(1,15).
7 ;TEMPORARY STORAGE OF 4 BITS OF SEGMENT DATA IN M(3,14).
8 ;KEYBOARD DEBOUNCE COUNTER (KBC) CONTAINED IN M(3,15).
9 ;SEVEN-SEGMENT DECODE ROM LOOKUP DATA CONTAINED IN PAGE
10 ;4, WORDS 0 - F.
11 ;KEYSTRAP DATA TIED TO D14-D12 LINES PLACED IN M(1,14)
12 ;THROUGH M(1,12).
13 ;EXIT TO KEYDECODE ROUTINE AFTER DEBOUNCING KEYSWITCH
14 ;CLOSURES WITH DIGIT VALUE IN M(1,15) AND G PORT DATA
15 ;IN A.
16          .SPACE      5          ;LEAVE 5 BLANK LINES ON LISTING
17          .PAGE      0
18          DIGIT      = 1,15      ;ASSIGN VALUE 1,15 TO "DIGIT"
19          STORE      = 3,14      ;ASSIGN VALUE 3,14 TO "STORE"
20          KBC        = 3,15      ;ASSIGN VALUE 3,15 TO "KBC"
21          CLRA
22 START:    JSR        CLRAM      ;FIRST INSTRUCTION MUST BE A "CLRA"
23          LBI        0,14        ;CLEAR ALL RAM
24 LDRAM:    CBA
25          XDS
26          ;LOAD DISPLAY REGISTER WITH NUMBERS
27          ;14 - 1
28          JP        LDRAM
28 DSPLY    OGI        15          ;SET ALL G PORTS HIGH
29          LBI        KBC
30          STII       15          ;POINT TO M(3,15)
31          ;15 TO KBC
31 DSP1:    LBI        0,14        ;NO, START DISPLAY AT DIGIT 14
32          DSP2:    CBA
33          XAD        DIGIT      ;DIGIT POSITION TO A
34          CLRA
35          AISC       4           ;STORE IN M(1,15)
36          LEI        0           ;SET A2 TO FLIP TO PAGE 1 FOR LOOKUP
37          LQID
38          LBI        DIGIT      ;BLANK SEGMENTS (RESET EN2)
39          LD         1           ;LOOKUP TABLE SEGMENT DATA TO Q
40          SKE
41          JMP        NODP       ;POINT TO DIGIT POSITION
42          CLRA
43          AISC       4           ;DIGIT POSITION TO A, POINT TO
44          JP        .-1         ;DECIMAL POINT POSITION DIGIT TO A
45          LBI        DIGIT      ;DECIMAL POINT = DIGIT POSITION?
46          LD         1           ;NO, RESET DECIMAL POINT BIT IN Q
47 DIGOUT:  LBI        DIGIT
48          LD
49          CAB
50          OBD
          ;DELAY 9 INSTR. CYCLE TIMES
          ;POINT TO DIGIT POSITION
          ;DIGIT POSITION TO A
          ;DIGIT POSITION TO BD
          ;OUTPUT DIGIT VALUE

```

Figure 6-1. DSPLY. SRC Source Code (Sheet 1 of 5)

## MONITOR PROGRAM LISTING (Continued)

```

51      LEI      4      ;OUTPUT SEGMENT DATA (SET EN2)
52      LBI      KBC    ;POINT TO KBC
53      ING
54      AISC     1      ;ALL G PORTS STILL HIGH (= 15)?
55      JMP      KEYDWN ;NO, JUMP TO "KEYDOWN" ROUTINE
56      CLRA
57      AISC     3      ;YES, DELAY 13 INSTR. CYCLE TIMES
58      JP       -1     ;BACK TO PREVIOUS INSTR. UNTIL SKIP
59      LBI      KBC
60      JMP      NRDY
61      .FORM
62      .PAGE     1      ;FORM FEED
63 ;WORDS 0-F EQUAL SEVEN-SEGMENT DECODE LOOKUP DATA TABLE
64 ;I(0) - I(7) = D.P., SG - SA
65 ;SENT UPON LOOKUP TO Q(7) - Q(0), RESPECTIVELY.
66 ;HEX VALUE FOR ASCII CHARACTERS 0 - 9,P,A,U,C,F,BLANK
67 ;PLACED IN SUCCESSIVE LOCATIONS BY ".WORD" DIRECTIVE
68      .SPACE   5      ;LEAVE 5 BLANK LINES ON LISTING
69      .WORD    X'FD    ;= 0      (7-SEGMENT DECODE HEX VALUES)
70      .WORD    X'6i    ;= i
71      .WORD    X'DB    ;= 2
72      .WORD    X'F3    ;= 3
73      .WORD    X'67    ;= 4
74      .WORD    X'B7    ;= 5
75      .WORD    X'3F    ;= 6
76      .WORD    X'E1    ;= 7
77      .WORD    X'FF    ;= 8
78      .WORD    X'E7    ;= 9
79      .WORD    X'CF    ;= P
80      .WORD    X'EF    ;= A
81      .WORD    X'7D    ;= U
82      .WORD    X'9D    ;= C
83      .WORD    X'8F    ;= F
84      .WORD    X'00    ;= BLANK
85 DEBOUN:
86      SKMBZ    3      ;UP BIT = 1?
87      JP       ALLUP  ;YES
88      SKMBZ    2      ;NO, NRB = 1?
89      JP       STR    ;YES, A = 15 SO STORE IT IN KBC
90 DECKBC: ADD
91 STR: X
92      SMB      3      ;SET UP BIT OF KBC
93      JMP      DSP1   ;DO DISPLAY LOOP OVER AGAIN
94 ALLUP: SKMBZ    2      ;NRB = 1?
95      JP       DECKBC ;YES, DECREMENT KBC (A = 15)
96      AISC     4      ;NO, SET KBC = 11
97      NOP
98      JP       STR    ;DEFEAT "AISC" SKIP
99 KEYDWN: LDD      DIGIT ;DIGIT POSITION TO A
100      AISC     4      ;DIGIT POSITION > 11 (STRAP DATA)?

```

Figure 6-1. DSPLY.SRC Source Code (Sheet 2 of 5)

## MONITOR PROGRAM LISTING (Continued)

```

101      JP      KBCSTST      ;NO
102      AISC    12          ;YES, RESTORE STRAP DIGIT VALUE
103      CAB
104      CLRA
105      AISC    1          ;1 TO BR(POINT TO STRAP DATA REG. 1)
106      XABR
107      ING
108      X
109      JMP     NRDY
110 KBCSTST: RMB     3          ;RESET UP BIT OF KBC
111      CLRA
112      AISC    8          ;DELAY 5 INSTR. CYCLE TIMES
113      JP      -1         ;REPEAT PREVIOUS INSTR. UNTIL SKIP
114      CLRA
115      SKE
116      JMP     NRDY
117      LEI     0          ;YES, BLANK SEGMENTS
118      ING
119      LBI     DIGIT      ;POINT TO DIGIT VALUE
120      JMP     KEYDEC     ;JUMP TO KEYDECODE ROUTINE
121      .FORM
122      .PAGE     2
123 CLRAM:   LBI     3,0
124 CLEAR:  CLRA
125      XIS
126      JP      CLEAR
127      XABR
128      AISC    15         ;REGISTER 0 CLEARED?
129      RET
130      XABR
131      JP      CLEAR
132 BLANK:  CLRA
133
134      AISC    15
135      XIS
136      JP      BLANK
137      RET
138      .FORM
139      .PAGE     4          ;FORM FEED
140 ;FOLLOWING CODE USES CONTENTS OF A AND M, KEYSWITCH COLUMN
141 ;AND ROW CLOSURE DATA, RESPECTIVELY, ON EXIT FROM DISPLAY
142 ;ROUTINE, TO ACCESS ROM POINTERS TO JUMP TO KEY1 - KEY16
143 ;DECODE ROUTINES. LABELS "KEY1" THROUGH "KEY16" MUST
144 ;BE LOCATED WITHIN PAGES 4 THROUGH 7.
145      .SPACE   5          ;FIVE BLANK LINES ON LISTING
146 KEYDEC:  COMP
147
148      JID
149
150      . = X'111          ;INDICATES KEY CLOSURE
                          ;JUMP TO KEYDECODE ROUTINE FOR
                          ;PARTICULAR KEY CLOSURE
                          ;MOVE ASSEMBLER LOCATION

```

Figure 6-1. DSPLY.SRC Source Code (Sheet 3 of 5)

## MONITOR PROGRAM LISTING (Continued)

```

151                                     ;COUNTER TO KEY1 ROM POINTER ADDRESS
152 .ADDR KEY1 ;PLACE KEY1 POINTER IN ADDRESS X'111
153 .ADDR KEY2 ;PLACE KEY2-KEY4 POINTERS IN NEXT
154 .ADDR KEY3 ;ROM LOCATIONS
155 .ADDR KEY4
156 . = X'121 ;MOVE TO KEY5 POINTER LOCATION
157 .ADDR KEY5
158 .ADDR KEY6
159 .ADDR KEY7
160 .ADDR KEY8
161 . = X'141 ;MOVE TO KEY 9 POINTER LOCATION
162 .ADDR KEY9 ;(PAGE 5)
163 .ADDR KEY10
164 .ADDR KEY11
165 .ADDR KEY12
166 . = X'181 ;MOVE TO KEY13 POINTER LOCATION
167 .ADDR KEY13 ;(PAGE 6)
168 .ADDR KEY14
169 .ADDR KEY15
170 .ADDR KEY16
171 KEY1: JMP ONE ;GO,D1 KEY
172 KEY2: JMP TWO ;GO,D2 KEY
173 KEY3: JMP THREE ;GO,D3 KEY
174 KEY4: JMP FOUR ;GO,D4 KEY
175 KEY5: JMP FIVE ;G1,D1 KEY
176 KEY6: JMP SIX ;G1,D2 KEY
177 KEY7: JMP SEVEN ;G1,D3 KEY
178 KEY8: JMP EIGHT ;G1,D4 KEY
179 KEY9: JMP NINE ;G2,D1 KEY
180 KEY10: JMP TEN ;G2,D2 KEY
181 KEY11: JMP ELEVEN ;G2,D3 KEY
182 KEY12: JMP TWELVE ;G2,D4 KEY
183 KEY13: JMP THIRTN ;G3,D1 KEY
184 KEY14: JMP FOURTN ;G3,D2 KEY
185 KEY15: JMP START ;G3,D3 KEY
186 KEY16: JMP START ;G3,D4 KEY
187 NRDY: LDD DIGIT ;DIGIT POSITION TO A
188 AISC 14 ;LAST DIGIT DONE (A = 1)?
189 JMP DEBOUN ;YES, JUMP TO DEBOUNCE ROUTINE (A = 15)
190 AISC 1 ;NO, DECREMENT DIGIT POSITION VALUE
191 LBI 0,0 ;POINT TO DISPLAY REGISTER 0
192 CAB ;DIGIT POSITION VALUE TO BD
193 CLRA
194 AISC 4 ;DELAY 9 INSTR. TIMES
195 JP .-1 ;REPEAT PREVIOUS INSTR. UNTIL SKIP
196 JMP DSP2 ;DISPLAY NEXT DIGIT
197 NODP: LBI STORE ;POINT TO M(2, 15)
198 CQMA ;SE-SG,D.P. TO A
199 X ;EXCHANGE INTO M (2,15)
200 RMB 0 ;RESETD.P. BIT (DECIMAL POINT OFF)

```

Figure 6-1. DSPLY .SRC Source Code (Sheet 4 of 5)

## MONITOR PROGRAM LISTING (Continued)

```

201          CAMQ          ;SEGMENT DATA BACK TO Q
202          JMP          DIGOUT
203
204          .FORM
205          .PAGE          7
206 ONE:      LBI          0,1
207 TWO:      LBI          0,2
208 THREE:    LBI          0,3
209 FOUR:     LBI          0,4
210 FIVE:     LBI          0,5
211 SIX:      LBI          0,6
212 SEVEN:    LBI          0,7
213 EIGHT:    LBI          0,8
214 NINE:     LBI          0,9
215 TEN:      LBI          0,10
216 ELEVEN   LBI          0,11
217 TWELVE   LBI          0,12
218 THIRTN:  LBI          0,13
219 FOURTN:  LBI          0,14
220          CBA
221          XAD          1,9          ;SAVE KEY NUMBER
222          LBI          0,0
223          JSR          BLANK       ;BLANK DISPLAY REGISTER
224          LBI          0,0
225          LDD          1,9          ;KEY NUMBER TO A
226          CAB
227          X
228          JMP          DSPLY       ;KEY NUMBER TO DISPLAY REGISTER
229          .END

```

Figure 6-1. DSPLY .SRC Source Code (Sheet 5 of 5)

## MONITOR PROGRAM LISTING (Continued)

```

1          .TITLE DSPLY, 'COP420 DISPLAY DEMO'
2          ;COP 420 DISPLAY/KEYBOARD DEBOUNCE/DECODE ROUTINE
3          ;DISPLAYS 14 BCD DIGITS CONTAINED IN M(0,14) THROUGH
4          ;M(0,1), HIGH-ORDER TO LOW-ORDER, RESPECTIVELY.
5          ;DECIMAL POINT POSITION VALUE CONTAINED IN M(0,15).
6          ;DIGIT POSITION VALUE CONTAINED IN M(1,15).
7          ;TEMPORARY STORAGE OF 4 BITS OF SEGMENT DATA IN M(3,14).
8          ;KEYBOARD DEBOUNCE COUNTER (KBC) CONTAINED IN M(3,15).
9          ;SEVEN-SEGMENT DECODE ROM LOOKUP DATA CONTAINED IN PAGE
10         ;4, WORDS 0 - F.
11         ;KEYSTRAP DATA TIED TO D14-D12 LINES PLACED IN M(1,14)
12         ;THROUGH M(1,12)
13         ;EXIT TO KEYDECODE ROUTINE AFTER DEBOUNCING KEYSWITCH
14         ;CLOSURES WITH DIGIT VALUE IN M(1,15) AND G PORT DATA
15         ;IN A
16         0005          .SPACE 5          ;LEAVE 5 BLANK LINES ON LISTING

17         0000          .PAGE          0
18         001F          DIGIT          = 1,15          ;ASSIGN VALUE 1,15 TO "DIGIT"
19         003E          STORE          = 3,14          ;ASSIGN VALUE 3,14 TO "STORE"
20         003F          KBC            = 3,15          ;ASSIGN VALUE 3,15 TO "KBC"
21         000 00          CLRA          ;FIRST INSTRUCTION MUST BE A "CLRA"
22         001 6880        START:        JSR          CLRAM          ;CLEAR ALL RAM
23         003 0D          LBI            0,14
24         004 4E          LDRAM:        CBA
25         005 07          XDS          ;LOAD DISPLAY REGISTER WITH NUMBERS
26                               ;14 - 1
27         006 C4          JP            LDRAM
28         007 335F        DSPLY:        OGI            15          ;SET ALL G PORTS HIGH
29         009 3E          LBI            KBC            ;POINT TO M(3,15)
30         00A 7F          STI            15            ;15 TO KBC
31         DSP1:
32         00R 0D          LBI            0,14          ;NO. START DISPLAY AT DIGIT 14
33         00C 4E          DSP2:        CBA            ;DIGIT POSITION TO A
34         00D 239F        XAD            DIGIT          ;STORE IN M(1,15)
35         00F 00          CLRA
36         010 54          AISC          4            ;SET A2 TO FLIP TO PAGE 1 FOR LOOKUP
37         011 3360        LEI            0            ;BLANK SEGMENTS (RESET EN2)
38         013 BF          LQID          ;LOOKUP TABLE SEGMENT DATA TO Q
39         014 1E          LBI            DIGIT          ;POINT TO DIGIT POSITION
40         015 15          LD            1            ;DIGIT POSITION TO A, POINT TO
41                               ;DECIMAL POINT POSITION DIGIT TO A
42         016 21          SKE          ;DECIMAL POINT = DIGIT POSITION?
43         017 61B2        JMP            NODP          ;NO, RESET DECIMAL POINT BIT IN Q
44         019 00          CLRA
45         01A 54          AISC          4

```

Figure 6-2. DISPLY.SRC Assembly Output Listing (Sheet 1 of 7)

# MONITOR PROGRAM LISTING (Continued)

46 01B DA	JP	.-1	;DELAY 9 INSTR. CYCLE TIMES
47 01C 1E	LBI	DIGIT	;POINT TO DIGIT POSITION
48 01D 05	LD		;DIGIT POSITION TO A
49 01E 50	CAB		;DIGIT POSITION TO BD
50 01F 333E	OBD		;OUTPUT DIGIT VALUE
51 021 3364	LEI	4	;OUTPUT SEGMENT DATA (SET EN2)
52 023 3E	LBI	KBC	;POINT TO KBC
53 024 332A	ING		;G PORTS TO A
54 026 51	AISC	1	;ALL G PORTS STILL HIGH (=15)?
55 027 605E	JMP	KEYDWN	;NO, JUMP TO "KEYDOWN" ROUTINE
56 029 00	CLRA		
57 02A 53	AISC	3	;YES, DELAY 13 INSTR. CYCLE TIMES
58 02B EA	JP	.-1	;BACK TO PREVIOUS INSTR. UNTIL SKIP
59 02C 3E	LBI	KBC	
60 02D 61A5	JMP	NRDY	

Figure 6-2. DSPLY.SRC Assembly Output Listing (Sheet 2 of 7)

## MONITOR PROGRAM LISTING (Continued)

```

61          .FORM          ;FORM FEED

62          0040          .PAGE 1
63          ;WORDS 0-F EQUAL SEVEN-SEGMENT DECODE LOOKUP DATA TABLE
64          ;I(0) - I(7) = D.P., SG - SA
65          ;SENT UPON LOOKUP TO Q(7) - Q(0), RESPECTIVELY.
66          ;HEX VALUE FOR ASCII CHARACTERS 0 - 9,P,A,U,C,F,BLANK
67          ;PLACED IN SUCCESSIVE LOCATIONS BY ". WORD" DIRECTIVE.
68          0005          .SPACE 5          ;LEAVE 5 BLANK LINES ON LISTING

69 040 FD          .WORD          X'FD          ;=0
70 041 61          .WORD          X'61          ;=1
71 042 DB          .WORD          X'DB          ;=2
72 043 F3          .WORD          X'F3          ;=3
73 044 67          .WORD          X'67          ;=4
74 045 B7          .WORD          X'B7          ;=5
75 046 3F          .WORD          X'3F          ;=6
76 047 E1          .WORD          X'E1          ;=7
77 048 FF          .WORD          X'FF          ;=8
78 049 E7          .WORD          X'E7          ;=9
79 04A CF          .WORD          X'CF          ;=P
80 04B EF          .WORD          X'EF          ;=A
81 04C 7D          .WORD          X'7D          ;=U
82 04D 9D          .WORD          X'9D          ;=C
83 04E 8F          .WORD          X'8F          ;=F
84 04F 00          .WORD          X'00          ;=BLANK
85          DEBOUN:
86 050 13          SKMBZ          3          ;UP BIT = 1?
87 051 D9          JP          ALLUP          ;YES
88 052 03          SKMBZ          2          ;NO, NRB = 1?
89 053 D5          JP          STR          ;YES, A = 15 SO STORE IT IN KBC
90 054 21          DECKBC: ADD          ;DECREMENT KBC
91 055 06          STR:          X          ;PLACE A IN KBC
92 056 4B          SMB          3          ;SET UP BIT OF KBC
93 057 600B        JMP          DSP1          ;DO DISPLAY LOOP OVER AGAIN
94 059 03          ALLUP:        SKMBZ          2          ;NRB = 1?
95 05A D4          JP          DECKBC          ;YES, DECREMENT KBC (A = 15)
96 05B 54          AISC          4          ;NO, SET KBC = 11
97 05C 44          NOP          ;DEFEAT "AISC" SKIP
98 05D D5          JP          STR
99 05E 231F        KEYDWN:       LDD          DIGIT          ;DIGIT POSITION TO A
100 060 54          AISC          4          ;DIGIT POSITION > 11 (STRAP DATA)?
101 061 EC          JP          KBCTST          ;NO
102 062 5C          AISC          12          ;YES, RESTORE STRAP DIGIT VALUE
103 063 50          CAB          ;STRAP DIGIT POSITION TO BD
104 064 00          CLRA
105 065 51          AISC          1
106 066 12          XABR          ;1 TO BR(POINT TO STRAP DATA REG. 1)
107 067 332A        ING          ;STRAP DATA TO A

```

Figure 6-2. DSPLY.SRC Assembly Output Listing (Sheet 3 of 7)



## MONITOR PROGRAM LISTING (Continued)

108 069 06		X		;PLACE IN APPROPRIATE DIGIT, REG. 1
109 06A 61A5		JMP	NRDY	
110 06C 43	KBCTST:	RMB	3	;RESET UP BIT OF KBC
111 06D 00		CLRA		
112 06E 58		AISC	8	;DELAY 5 INSTR. CYCLE TIMES
113 06F EE		JP	.-1	;REPEAT PREVIOUS INSTR. UNTIL SKIP
114 070 00		CLRA		;0 TO A
115 071 21		SKE		;KBC = 0?
116 072 61A5		JMP	NRDY	;NO
117 074 3360		LEI	0	;YES, BLANK SEGMENTS
118 076 332A		ING		;G PORTS TO A
119 078 1E		LBI	DIGIT	;POINT TO DIGIT VALUE
120 079 6100		JMP	KEYDEC	;JUMP TO KEYDECODE ROUTINE
121		.FORM		
122 0080		.PAGE	2	
123 080 3F	CLRAM:	LBI	3,0	
124 081 00	CLEAR:	CLRA		
125 082 04		XIS		
126 083 81		JP	CLEAR	
127 084 12		XABR		
128 085 5F		AISC	15	;REGISTER 0 CLEARED?
129 086 48		RET		;YES, RETURN
130 087 12		XABR		;NO, BR -1 TO BR
131 088 81		JP	CLEAR	
132 089 00	BLANK:	CLRA		;PLACE "F"'S (DISPLAY BLANKS) IN A
133				;RAM REGISTER
134 08A 5F		AISC	15	
135 08B 04		XIS		
136 08C 89		JP	BLANK	
137 08D 48		RET		

Figure 6-2. DSPLY.SRC Assembly Output Listing (Sheet 4 of 7)

## MONITOR PROGRAM LISTING (Continued)

```

138          .FORM          ;FORM FEED

139      0100          .PAGE 4
140          ;FOLLOWING CODE USES CONTENTS OF A AND M, KEYSWITCH COLUMN
141          ;AND ROW CLOSURE DATA, RESPECTIVELY, ON EXIT FROM DISPLAY
142          ;ROUTINE, TO ACCESS ROM POINTERS TO JUMP TO KEY1 - KEY16
143          ;DECODE ROUTINES. LABELS "KEY1" THROUGH "KEY16" MUST
144          ;BE LOCATED WITHIN PAGES 4 THROUGH 7.
145      0005          .SPACE 5          ;FIVE BLANK LINES ON LISTING

146      100 40          KEYDEC:  COMP          ;COMPLEMENT A SO THAT BIT = 1
147          ;INDICATES KEY CLOSURE
148      101 FF          JID          ;JUMP TO KEYDECODE ROUTINE FOR
149          ;PARTICULAR KEY CLOSURE
150      0111          . = X'111          ;MOVE ASSEMBLER LOCATION
151          ;COUNTER TO KEY1 ROM POINTER ADDRESS
152      111 85          .ADDR  KEY1          ;PLACE KEY1 POINTER IN ADDRESS X'111
153      112 87          .ADDR  KEY2          ;PLACE KEY2-KEY4 POINTERS IN NEXT
154      113 89          .ADDR  KEY3          ;ROM LOCATIONS
155      114 8B          .ADDR  KEY4
156      0121          . = X'121          ;MOVE TO KEYS POINTER LOCATION
157      121 8D          .ADDR  KEY5
158      122 8F          .ADDR  KEY6
159      123 91          .ADDR  KEY7
160      124 93          .ADDR  KEY8
161      0141          . = X'141          ;MOVE TO KEY9 POINTER LOCATION
162      141 95          .ADDR  KEY9          ;(PAGE 5)
163      142 97          .ADDR  KEY10
164      143 99          .ADDR  KEY11
165      144 9B          .ADDR  KEY12
166      0181          . = X'181          ;MOVE TO KEY13 POINTER LOCATION
167      181 9D          .ADDR  KEY13          ;(PAGE 6)
168      182 9F          .ADDR  KEY14
169      183 A1          .ADDR  KEY15
170      184 A3          .ADDR  KEY16

171      185 61C0      KEY1:  JMP  ONE          ;G0,D1 KEY
172      187 61C2      KEY2:  JMP  TWO          ;G0,D2 KEY
173      189 61C4      KEY3:  JMP  THREE        ;G0,D3 KEY
174      18B 61C6      KEY4:  JMP  FOUR         ;G0,D4 KEY
175      18D 61C8      KEY5:  JMP  FIVE        ;G1,D1 KEY
176      18F 61CA      KEY6:  JMP  SIX         ;G1,D2 KEY
177      191 61CC      KEY7:  JMP  SEVEN       ;G1,D3 KEY
178      193 61CE      KEY8:  JMP  EIGHT      ;G1,D4 KEY
179      195 61D0      KEY9:  JMP  NINE       ;G2,D1 KEY
180      197 61D1      KEY10: JMP  TEN        ;G2,D2 KEY
181      199 61D2      KEY11: JMP  ELEVEN     ;G2,D3 KEY
182      19B 61D3      KEY12: JMP  TWELVE    ;G2,D4 KEY
183      19D 61D4      KEY13: JMP  THIRTN    ;G3,D1 KEY
184      19F 61D5      KEY14: JMP  FOURTN    ;G3,D2 KEY

```

Figure 6-2. DSPLY .SRC Assembly Output Listing (Sheet 5 of 7)

## MONITOR PROGRAM LISTING (Continued)

```

185 1A1 6001      KEY15:  JMP      START      ;G3,D3 KEY
186 1A3 6001      KEY16:  JMP      START      ;G3,D4 KEY
187 1A5 231F      NRDY:   LDD      DIGIT      ;DIGIT POSITION TO A
188 1A7 5E        AISC    14          ;LAST DIGIT DONE (A = 1)?
189 1A8 6050      JMP     DEBOUN     ;YES, JUMP TO DEBOUNCE ROUTINE (A = 15)
190 1AA 51        AISC    1          ;NO, DECREMENT DIGIT POSITION VALUE
191 1AB 0F        LBI     0,0        ;POINT TO DISPLAY REGISTER 0
192 1AC 50        CAB                    ;DIGIT POSITION VALUE TO BD
193 1AD 00        CLRA                    ;
194 1AE 54        AISC    4          ;DELAY 9 INSTR. TIMES
195 1AF EE        JP      .-1        ;REPEAT PREVIOUS INSTR. UNTIL SKIP
196 1B0 600C      JMP     DSP2       ;DISPLAY NEXT DIGIT
197 1B2 3D        NODP:  LBI     STORE     ;POINT TO M(2,15)
198 1B3 332C      CQMA                    ;SE-SG,D.P. TO A
199 1B5 06        X                        ;EXCHANGE INTO M(2,15)
200 1B6 4C        RMB     0          ;RESETD.P. BIT (DECIMAL POINT OFF)
201 1B7 333C      CAMQ                    ;SEGMENT DATA BACK TO Q
202 1B9 601C      JMP     DIGOUT     ;
203

204              .FORM

205              .PAGE      7
206 1C0 3381      ONE:    LBI     0,1
207 1C2 3382      TWO:   LBI     0,2
208 1C4 3383      THREE: LBI     0,3
209 1C6 3384      FOUR:  LBI     0,4
210 1C8 3385      FIVE:  LBI     0,5
211 1CA 3386      SIX:   LBI     0,6
212 1CC 3387      SEVEN: LBI     0,7
213 1CE 3388      EIGHT: LBI     0,8
214 1D0 08        NINE:  LBI     0,9
215 1D1 09        TEN:   LBI     0,10
216 1D2 0A        ELEVEN: LBI     0,11
217 1D3 0B        TWELVE LBI     0,12
218 1D4 0C        THIRTN: LBI     0,13
219 1D5 0D        FOURTN: LBI     0,14
220 1D6 4E        CBA
221 1D7 2399      XAD     1,9        ;SAVE KEY NUMBER
222 1D9 0F        LBI     0,0
223 1DA 6889      JSR     BLANK     ;BLANK DISPLAY REGISTER
224 1DC 0F        LBI     0,0
225 1DD 2319      LDD     1,9        ;KEY NUMBER TO A
226 1DF 50        CAB
227 1E0 06        X                        ;KEY NUMBER TO DISPLAY REGISTER
228 1E1 6007      JMP     DSPLY
229              .END

```

Figure 6-2. DSPLY.SRC Assembly Output Listing (Sheet 6 of 7)

**MONITOR PROGRAM LISTING** (Continued)

ALLUP	0059	BLANK	0089	CLEAR	0081	CLRAM	0080
DEBOUN	0050	DECKBC	0054	DIGIT	001F	DIGOUT	001C
DSP1	000B	DSP2	000C	DSPLY	0007	EIGHT	01CE
ELEVEN	01D2	FIVE	01C8	FOUR	01C6	FOURTN	01D5
KBC	003F	KBCTST	006C	KEY1	0185	KEY10	0197
KEY11	0199	KEY12	019B	KEY13	019D	KEY14	019F
KEY15	01A1	KEY16	01A3	KEY2	0187	KEY3	0189
KEY4	018B	KEY5	018D	KEY6	018F	KEY7	0191
KEY8	0193	KEY9	0195	KEYDEC	0100	KEYDWN	005E
LDRAM	0004	NINE	01D0	NODP	01B2	NRDY	01A5
ONE	01C0	SEVEN	01CC	SIX	01CA	START	0001
STORE	003E	STR	0055	TEN	01D1	THIRTN	01D4
THREE	01C4	TWELVE	01D3	TWO	01C2		

NO ERROR LINES

227 ROM WORDS USED

SOURCE CHECKSUM = D7F0

INPUT FILE 13:DSPLY.SRC

LISTING FILE 13:DSPLY.LST

Figure 6-2. DSPLY.SRC Assembly Output Listing (Sheet 7 of 7)

# COPS Monitor and Debugger (COPMON)

## 7.1 COPMON (COP Monitor)

COPMON is a PDS system program which contains an extensive set of debugging commands. This chapter discusses the format of the COPMON commands and their use in debugging programs and hardware.

COPMON allows the user to interrupt the flow of a COPS program as it is being executed on a prototype system. The interruption is directly caused by one of several events, all under user control. For instance, the interruption may be caused by the COPS chip performing an instruction fetch from a predetermined point in the program called a breakpoint. Once the flow has been interrupted, the COPS registers can be examined and modified. COPMON also allows the user to examine the trace of the program flow for the last 256 instruction cycles, either before or after a specified condition was met. This is called a trace. Possible conditions for a breakpoint or trace may be the program encountering a specified address (address), the next value of the program counter (immediate), or any combination of two external events on the Emulation Board called EXT1 and EXT2.

The TRACE command allows the user to specify the conditions that will initiate the trace and how many steps prior to meeting that condition will be traced. The GO command then arms the trace and executes the program. After a trace has been completed, the operator may examine the trace data with a TYPE command or search for an address in the trace memory with the expected sequence of instructions, deviations resulting from incorrect operation are easily found.

To speed operation, COPMON allows the operator to specify the information that will be printed out when a breakpoint occurs and to single-step operations. This is done with the AUTOPRINT command, especially useful during single-step operation. The COPS registers and RAM locations can also be examined and modified directly with MODIFY. The program in shared memory can be changed with ALTER or PUT.

Another major function available on COPMON is the Time command. This can be used to determine the time, in milliseconds, between two specified trigger conditions. (A trigger condition can be an address or any combination of the external event lines EXT1 and EXT2.)

## 7.2 Console Operation

To call COPMON from the console, the user types in the @ command, getting the following response:

```
>@COPMON
COPMON, REV: X, (DATE)
CHIP NUMBER (DEFAULT = 420)? XXX
```

The operator must enter a chip number from Table 7-1 in response to the system query. The chip number is used by COPMON to select the correct instruction subset, memory size, and register size. If no number is entered after the chip number prompt, COPMON defaults to the COP420 number. The chip number may also be changed later with the CHIP command. After

the operator responds to the initial chip number prompt, COPMON responds with the COPMON prompt symbol, C>.

Example:

```
CHIP NUMBER (DEFAULT = 420)? 444
CHIP BEING EMULATED: COP444
C>
```

COPMON responds with the prompt after completing the execution of each command.

The following general rules apply to the console commands:

1. Numbers — COPMON syntax uses both decimal and hexadecimal numbers (see Table 7-3). Input from the user is treated as decimal or hexadecimal depending on what COPMON is expecting. If COPMON expects a decimal number it assumes that the user will enter a decimal number. Hexadecimal numbers do not require a leading zero; however, they do no harm since they are ignored. F3 is a valid hexadecimal number. The usual conventions for hexadecimal, an H at the end of a hexadecimal number (3FH) or an X at the beginning of a hexadecimal number (X'1F) are illegal.
2. Console Output — Console output of COPMON is normally sent to the CRT. The output of any one command may be directed to the printer by appending "\*PR" to the end of the command. (The "\*PR" must be immediately followed by a carriage return.)  
Example: C>STATUS \*PR  
The status now appears on the printer, instead of the CRT.  
Console output (whether to the CRT or line printer), may be interrupted at any time by pressing any key. Asterisks (\*\*\*\*\*) will be printed to indicate this.
3. Disk Files — The LOAD, COMPARE and SAVE commands use disk files. The default extension assumed is ".LM".

## 7.2.1 Dual Processor Emulation

Users of the dual processor COPS chips (COP2440, COP2441, and COP2442) should note the following points before attempting emulation.

1. The two processors are referred to as the X and Y processors. Processor X starts execution at address 0H on power-on, processor Y at address 401H.
2. COPMON makes sure that the two processors are always synchronized, that is, they execute instructions in the same order as they would if there were no breakpoints. While single-stepping, it is sometimes necessary for one processor to execute two or more instructions before the other executes any (for example, if one processor is breakpointed on a skipped 2-byte instruction).

Table 7-1. Valid Chip Numbers

Chip #	Memory Size	RAM Register Address	RAM Digit Address
410/411	0-1FFH	0H-3H	0,9H-0FH
420/421/422	0-3FFH	0H-3H	0*H-0FH
444/445	0-7FFH	0H-7H	0H-0FH
440/441/442	0-7FFH	0H-9H	0H-0FH
2440/2441/ 2442	0-7FFH	0H-9H	0H-0FH

Note: One of these numbers must be entered into the computer in response to the query for CHIP NUMBER?. If no number is entered, COPMON will use the default chip number 420.

It is possible for this synchronism to be lost, though it should not happen under normal circumstances.

When the Program Counters are printed, an asterisk (\*) is used to mark the PC of the processor which will execute next.

- The hardware places some restrictions on triggering from a reset state. The target board synchronizes when processor Y sends out address 401H. If the trigger condition becomes valid before this, correct synchronization is uncertain. For example, if a TRACE IMMEDIATE is performed from RESET, processors X and Y may get interchanged: i.e., processor X will be displayed on the right-hand side of the screen, instead of the left-hand side as usual. There is a 50-50 chance of this happening.

This uncertainty also exists if an External Event condition is used for TRACE, BREAKPOINT or TIME operations starting from RESET and the condition is valid before address 401H appears.

AS FAR AS POSSIBLE, SUCH UNCERTAIN TRIGGERING CONDITIONS SHOULD BE AVOIDED. IF SUCH AN OPERATION HAS TO BE PERFORMED, THE COP CHIP SHOULD BE RESET AFTER THE OPERATION, SINCE FURTHER EMULATION MAY BE INCORRECT.

- COPMON operates in three basic modes, referred to as the DUAL, X-only and Y-only modes. The DUAL mode is the default, 'normal' mode of operation. The X-only and Y-only modes make it simpler to concentrate on the behavior of one particular processor and temporarily ignore the other. Refer to the 'SET PROCMODE' (Section 7.3.22) command for details.

### 7.3 COPMON Console Commands

The COPMON console commands are summarized in Table 7-2 and are described in detail here. Command options are defined in Table 7-3.

#### 7.3.1 ALTER SHARED MEMORY Command

Syntax: ALTER [addr][, [value]] . . .

Alters the contents of consecutive shared memory locations to the specified hexadecimal values beginning at the specified address. Consecutive commas will increment the current address pointer, leaving the data at these locations unaltered. If no address is specified, then

command begins at the last altered or listed location (see LIST command). If two or more values separated by spaces are given for <value>, the last of these values will be the one stored. The alterable range of shared memory is determined by the chip number. The COP chip is reset if it was running.

Example:

```
C>A 1CF,D0,,D1 ← Places D0 in location 1CF,
leaves 1D0 unchanged, and places D1 in location
1D1.
```

#### 7.3.2 AUTOPRINT Command

Syntax: AUTOPRINT [<print opt>][, <print opt>] . . .

Specifies the information that will be printed when the COPS chip encounters a breakpoint, is single-stepped, or executes a trace. Table 7-3 lists all of the allowable print options. The default value is ALL which sets all of the applicable options on, except S and ST. Some of the print options are only valid for breakpoints and single-steps; others are valid for trace operations. A "'PR'" entered at the end of the line will cause the autoprnt output to go to the printer instead of the console. The 16-digit contents of any specified RAM register will be printed, left to right, most-significant digit to least-significant digit.

Example:

```
C>AU A, P
```

Causes the contents of the accumulator and the program counter to be printed after each breakpoint and single-step operation.

If it is desired to modify the current list of print options, a "+" or "-" may be placed in front of the list of options. In this case, ALL may not be used as a print option.

Example:

```
C>AU A,P ←Accumulator and program counter put
in print option list.
```

```
C>AU +M2 ←Now memory register 2 is also printed
along with the accumulator and program counter.
```

If no <print opt>'s are specified, the autoprnt feature is turned off.

Example:

```
C>AU ← AUTOPRINT off
```

COP2440, COP2441, COP2442 users should refer to the 'SET PROCMODE' command (Section 7.3.21) for changes in <print opt>'s with the default processor setting.

#### 7.3.3 BREAKPOINT Command

Syntax: BREAKPOINT [<cond>[/<cond>]] . . .  
[,<occur#>][,<gopt>]]

Sets the breakpoint enable flag and establishes the conditions that will cause breakpoints to occur. Up to ten conditions can be specified, but only the first will be monitored. When that condition is satisfied and a breakpoint executed, the list of conditions is rotated so the next condition on the list becomes the one being monitored. If the BREAKPOINT command is entered with no conditions specified, all previous conditions are

retained. If the BREAKPOINT command is entered with one or more conditions, all of the previous conditions are cleared and replaced by the new ones contained in the command string. If the occurrence number is not specified, the system defaults to the last specified value. If <gopt> is specified, the breakpoint operation occurs repeatedly on successive conditions in a circular list. This continues until a break is received from the console. When the breakpoint occurs, the data specified earlier by the AUTOPRINT command is printed out to provide the operator with a snapshot of the pertinent data during the COPS program execution.

During a breakpoint, the system automatically does a trace with a prior count of 240. This information about the 240 cycles prior to the breakpoint may be printed using the TYPE command. Locations corresponding to the breakpointed state of the chip are displayed as asterisks ("\*\*\*\*\*").

The BREAKPOINT command sets the breakpoint enable flag but does not actually initiate the breakpoint. This is done by the next GO command which initiates program execution. Since the breakpoint operation uses shared memory, if the operator is running from programs contained in PROMs on the emulator board, the shared memory must contain the same data as the PROMs.

Example:

```
C>B 2/35//EVX1/26, 4, G
BREAKPOINT ENABLED
A:2 A:35 IMED EVX1 A:26 OCCUR:4 GO:Y
```

Break flag is enabled, the next GO will cause successive breakpoints on the fourth occurrence of each of the five conditions, circling through the list until interrupted.

COP2440, COP2441, COP2442 users should refer to the 'SET PROCMODE' (Section 7.3.21) command for changes in <cond> with the default processor setting. Also, during a breakpoint, both processors are traced, even if the mode is X-only or Y-only.

### 7.3.4 CLEAR Command

Syntax: CLEAR

Clears the breakpoint enable, trace enable, and time enable flags. The conditions associated with each of these functions remain unchanged.

Example:

```
C>C
BREAKPOINT, TRACE, AND TIME DISABLED
```

### 7.3.5 CHIP Command

Syntax: CHIP <chip#>

Changes and displays the current chip number. Since the chip number determines the memory and register size, this must be done prior to emulating a COPS chip. See Table 7-1. If no chip is specified, the current chip number is displayed.

Example:

```
C>CH 444
CHIP BEING EMULATED: COP444
```

Example:

```
C>CH
CHIP BEING EMULATED: COP444
```

If the chip being emulated is a COP410 or a COP411, COPMON will respond with another query:

```
ROMLESS PART (DEFAULT = 401)?
```

The operator must enter one of the following: 401, 402 or 404 depending on which ROMless part is being used on the emulator board (COP401L, COP402 or COP404L).

Example:

```
CHIP NUMBER (DEFAULT = 420)? 411
ROMLESS PART (DEFAULT = 401)? 402
CHIP BEING EMULATED: COP411
ROMLESS PART BEING USED: COP402
C>
```

### 7.3.6 COMPARE Command

Syntax: COMPARE <filename>

Checks the load module on disk against shared memory. Each pair of values that does not compare is displayed. This continues until either the entire file has been examined or a break is received from the console. The COP chip is reset.

Example:

```
C>CO DEMO
003 S:00 F:3C 057 S:8A F:B3 ← S: indicates
shared memory; F: indicates a disk file; 003: indi-
cates an address.
```

Note: Only those shared memory locations which are defined in the load module are compared.

### 7.3.7 DEPOSIT Command

Syntax: DEPPOSIT <value>, <addr range>

Puts the specified value into each location of the specified address range. If the COP chip is running, it will be reset.

Example:

```
C>D F6, 11/1E ← F6 is put in locations 11 through
1E of shared memory.
```

### 7.3.8 FIND Command

Syntax: FIND <value>[, <addr range>[, <mask>]]

Searches shared memory for the specified value and each occurrence is printed out. If the mask option is present, each shared memory byte is ANDed with the value of <mask> before it is tested. This allows the user to search out specific portions of bytes. If the mask option is not specified, it defaults to 0FFH. Each occurrence of value is printed on the console until the search is done or it is interrupted from the console. If the COP chip is running it will be reset.

Example:

```
C>F 8E, 200/3FF
2CC:8E 2B0:8E 3FF:8E
FIND DONE
```

If the <value> typed in is three characters or more, a 2-byte search is performed. This is useful for locating 2-byte instructions. In this case, the mask defaults to 0FFFFH.

Example:

```
C>F 6310, 100/3FF
275:6310 372:6310
FIND DONE (2 BYTE)
```

### 7.3.9 GO Command

Syntax: GO [<addr>]

GO [<addr>][,<addr>] ← *Dual processor only, see note below.*

Goes to a specified address and begins executing the program there. The details of exactly how this is done vary somewhat depending on the status of the chip and the breakpoint and trace enable flags. Generally speaking, a breakpoint will be initiated if the breakpoint enable flag is set, a trace will be done if the trace enable flag is set, a time operation will be done if the time enable flag is set, and the chip will be started in a normal manner if no flag is set. See Table 7-4. Breakpoint and trace flags remain unchanged after the GO command. For example, if the breakpoint flag is enabled, the first condition on the list is EV0X, the autoprint options are B,P, and <gopt> is not set, the following sequence will occur:

Example:

```
C>G
BREAKPOINTED ON EV0X AT A:xxx
B:01 P:xxx
```

xxx indicates the address at which EV0X occurred. A similar message would appear if trace were enabled instead of breakpoint.

Note: For COP2440, COP2441, and COP2442 users: Two addresses can be specified when emulating dual-processor COPS.

<addr> = address for processor X  
<addr> = address for processor Y

If the processor mode is X-only or Y-only (see 'SET PROCMODE' command), and a single address is specified, it is assumed to refer to the default processor.

Example:

```
C>SET PR Y ← Set processor Y as default
C>G 58 ← Will start processor Y at address 58
C>G 27,439 ← Start processor X at 27, processor Y at 439.
```

### 7.3.10 HELP Command

Syntax: HELP

The HELP command causes a summary of the COP-MON commands to be printed on the console.

### 7.3.11 LIST Command

Syntax: LIST [<addr range>,<addr range>]. . . ]

Lists the contents of shared memory across the specified address ranges. Each range printed begins at the next lower multiple of 10H. If <addr range> is just one

value, only the contents of that location are printed. If no address range is specified, 256 locations are listed starting from the multiple of 10H below the current address. The current address is the last address printed or altered. Subsequent LIST commands with no operands will list the next 256 locations. The COPS chip is reset only if it was running when the LIST command was issued.

Example:

```
C>L 4/8
000 00 C2 00 F2 03 29 76 AA D0
```

### 7.3.12 LOAD Command

Syntax: LOAD <filename>[O]

Loads the specified load module into shared memory. If the optional "O" (for 'Overlay') is present in the command string, shared memory will not be cleared out first. LOAD automatically resets the COPS chip.

Example:

```
C>LO DEMO
FINISHED LOADING
```

### 7.3.13 MODIFY Command

Syntax: MODIFY <print opt>,<value>[,<value>]. . .

Changes the registers on the COPS chip. Since these registers include the I/O ports as well as the general purpose registers and RAM registers, the MODIFY command can be used to debug a hardware prototype system prior to the prototype software being completed. Each MODIFY command line is used to change a single register on the chip. The MODIFY command is valid only while breakpointed.

Example:

```
C>B 1
BRKPT ENABLED
A:001 OCCUR 1 GO:IN
C>R
CHIP IS RESET
C>G
BREAKPOINTED ON A:001 AT A:001
C>M M0, 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F ← This command sets memory register 0 digit 0 to 0, memory register 0 digit 1 to 1, etc.
C>M M15,5,6,7,8. . . ← This command sets memory register 1 digit 5 to 5, etc.
C>M E,4 ← This command loads the E register with 4 (enable Q register to L bus).
C>M Q,AA ← This command in conjunction with the previous command loads the Q register with AA and thus puts AA on the L bus.
C>M D,B ← This command puts a HEX B on the D port. Bits 0, 1, 3 are high and bit 2 is low.
C>M B,3D ← This command sets the B register to RAM address 3,13.
```

COP2440, COP2441, and COP2442 users should refer to the 'SET PROCMODE' command (Section 7.3.22) for changes in <print opt>'s with the default processor setting.



### 7.3.14 NEXT Command

Syntax: NEXT [<gopt>]

NEXT [<gopt>]|X[,<gopt>]  
|Y[,<gopt>] ← *Dual processor only. See 'SET PROCMODE' command.*

This command is identical to the SINGLESTEP command (see Section 7.3.17), except at a JSR or JSRP instruction, where it will set a breakpoint at the instruction immediately after the JSR/JSRP and breakpoint there, after executing the subroutine in real-time.

### 7.3.15 PUT Command

Syntax: PUT [<addr>][,<instruct>][,<instruct>]...

Replaces the contents of shared memory, beginning at the address specified, with the opcodes of the specified instruction mnemonics. If no address is given, placement begins at the current address. This command resets the COPS chip if it is running. Instruction opcodes may be directly specified in the operand field. Instructions with double operands may only be specified in hexadecimal format and, unlike the assembler format, double operands may not be separated by commas (e.g., LBI 23 is OK; LBI 2,3 is not allowed).

Example:

```
C>P 130, CLRA, AISC 5, LBI 39
C>
```

### 7.3.16 RESET Command

Syntax: RESET

Reset the COPS chip and sets the reset flag, which in turn determines the operation of the GO command. See Table 7-4.

Example:

```
C>R
CHIP IS RESET
```

### 7.3.17 SINGLESTEP Command

Syntax: SINGLESTEP [<gopt>]

SINGLESTEP [<gopt>]|X[,<gopt>]  
|Y[,<gopt>] ← *Dual processor only. See 'SET PROCMODE' command.*

Performs a breakpoint on the next instruction. If the COPS chip is reset, it breakpoints at address 1. If it has already breakpointed, it steps one instruction. After each single-step, information specified in the AUTOPRINT command is printed. If <gopt> is included, it will automatically step and print data until interrupted by the console.

If the COP chip is breakpointed, a carriage return is identical to single-step without <gopt>.

Example:

```
C>S G ← Go immediately after printing.
(Step)
A:0 P:10 51 AISC 1
(Step)
A:1 P:11 53 AISC 3
```

### 7.3.18 SAVE Command

Syntax: SAVE <filename>

Saves the contents of shared memory in the specified file. All of shared memory, from address 0 to the maximum address of the chip being emulated, is saved. Shared memory itself is unchanged. This file may later be loaded back into shared memory using the LOAD command. The COP chip will be automatically reset. The saved program cannot be used in MASKTR to generate a transmittal file.

Example:

```
C>SA MYPROG
SAVED MYPROG
```

### 7.3.19 SEARCH Command

Syntax: SEARCH <addr>

Searches the trace memory for the specified address. Each occurrence is displayed and it searches until finished or interrupted by the console. Each line of output from the SEARCH command and the TYPE (trace memory) command contains the following information, from left to right:

1. Trace Memory Location.
2. Location relative to TRACE condition location.
3. Program Counter.
4. Skip Indication.
5. Value of external event inputs E4-E1, left to right.

Example:

```
C>SE 2FE
0 0 A:2FE SKIP E:1111
8 8 A:2FE E:1101
SEARCH DONE
```

### 7.3.20 SET Command

Syntax: SET SIOMODE {Y|N}

SET STACKMODE {Y|N}

Turns the SIOMODE and STACKMODE flags on and off. The SIO register will be dumped during breakpoint and can be modified only if SIOMODE is on. Similarly, if STACKMODE is on, the stack will be dumped and displayed during breakpoint and single-step. The stack may also be modified.

There is one limitation in using STACKMODE. If the COP is breakpointed in an interrupt routine and STACKMODE is ON, the interrupt skip status flag in the COP chip may be lost. It cannot be restored. (If lost, a message will be printed.) This limitation does not apply to the COP440, COP441, COP442, COP2440, COP2441, COP2442 and hence, for these chips, the default is STACKMODE ON.

Use of the SET command will automatically reset the COP chip and set AUTOPRINT to ALL.

Example:

```
C>SET ST Y
STACKMODE: Y
```

### 7.3.21 SET PROCMODE Command (COP2440, COP2441 and COP2442 only)

Syntax: SET PROCMODE {X|Y|D}

Sets the default processor mode for dual processor emulation.

The effects of setting a particular mode are best seen by example.

#### 1. BREAKPOINT, TRACE and TIME <cond>s.

A hexadecimal address by itself refers to the default processor.

Example:

```
C>SET PR D ← Set 'DUAL' mode.
C>B 23 ← Breakpoint on address 23 of either processor X.
C>B 23-X ← Breakpoint on address 23 of processor.
C>SET PR X ← Set 'X-only' mode (i.e., default is X).
C>B 234 ← Breakpoint on address 234 of processor X.
C>TR 23-Y ← Trace on address 23 of processor Y.
C>TR 23-D ← Trace on address 23 of either processor.
```

The default processor setting has no effect on external event or immediate triggering.

Example:

```
C>TR EVX1
```

This will initiate a trace when external event 1 = 1, regardless of the default processor setting and regardless of the processor cycle during which the event is detected.

#### 2. AUTO PRINT, TYPE and MODIFY <print opt>s.

Example:

```
C>SET PR D ← Set 'DUAL' mode.
C>AU AX,CY ← Will print AX and CY.
C>MO A,3 ← Is ambiguous (Modify AX or AY?).
C>SET PR Y ← Set 'Y-only' mode (i.e., default is Y).
C>AU ALL ← Will print all RAM I/O registers, and all processor Y registers (i.e., AY,CY, etc.).
C>AU A,BX,C ← Will print AY,BX,CY.
C>MO A,3 ← Will modify AY to 3.
```

#### 3. SINGLESTEP and NEXT operations.

Syntax: SINGLESTEP [<gopt>]|X[,<gopt>]  
|Y[,<gopt>]  
(or NEXT)

Example:

```
C>SET PR D ← Set 'DUAL' mode.
C>S ← Single-step on processor which is to execute next.
C>S G ← Single-step continuously on alternate processors.
C>SET PR X ← Set 'X-only' mode (i.e., default is X).
C>N ← Do a 'NEXT' on processor X.
```

```
C>S Y ← Single-step on processor Y.
```

```
C>S G ← Single-step continuously on processor Y.
```

#### 4. GO operation.

Refer to 'GO' command description, Section 7.3.10.

As with the other SET commands, the SET PROCMODE command will reset the COPS chip and restore default AUTO PRINT conditions. In addition, it will set BREAKPOINT, TRACE, and TIME conditions to their default values.

### 7.3.22 SHARED MEMORY Command

Syntax: SHARED MEMORY {Y|N}

The command allows the operator to specify whether the COPS chip runs from shared memory or the PROMs on the emulator board. If "Y" is entered, the chip will run from shared memory. If "N" is entered, the chip will run from the PROMs. The chip is automatically reset by this command.

Example:

```
C>SH Y
SHARED MEMORY MODE
C>SH N
PROM MODE
C>
```

### 7.3.23 STATUS Command

Syntax: STATUS

This command causes the status of the COPS chip and various other internal conditions to be printed.

Examples:

```
C>ST
CHIP BEING EMULATED: COP 420
CHIP IO RESET
BREAKPOINT, TRACE AND TIME DISABLED
SHARED MEMORY MODE
NO UNASSEMBLY
SIO REG MODE: N
STACK MODE: N
BRKPT CONDITIONS:
A:005 OCCUR: 1 GO:N
TRACE CONDITIONS:
EVX1 OCCUR: 1 PRIOR:0 GO:N
TIME CONDITIONS:
A:001 OCCUR:1 A:237 OCCUR:2 GO:Y
```

### 7.3.24 TIME Command

Syntax: TIME [<cond1>[,<occur1>]]/[<cond2>[,<occur2>][,<gopt>]]]

Sets and prints the conditions which control the time measurement. The timer is started when the first set of conditions is met and the timer is stopped when the second set of conditions is met. The second set of conditions is invoked only after the first set of conditions is satisfied, and it is looked for from that time. If only cond1 is specified, cond2 is set to cond1 and occurrences are both set to the last value of occur1. If only cond1 and occur1 are specified, cond2 is set the same as cond1 and

occur2 is set the same as occur1. If cond1 and cond2 and specified, occur1 and occur2 are left at their previous values.

The time is reported in milliseconds. The limits on the TIME command are: the time between the events must be greater than 500  $\mu$ s and less than 2 hours. If the time is less than 500  $\mu$ s, the events may not be recognized, or if they are recognized, the time reported could be wrong. If the time is greater than 2 hours, a timer overflow message will be printed. The resolution of the TIME command is  $\pm 100 \mu$ s.

As in the TRACE command, the TIME command is not initiated until a GO command is issued. The TIME, TRACE, and BREAKPOINT commands are mutually exclusive.

COP2440, COP2441, COP2442 users should refer to the 'SET PROCMODE' command (Section 7.3.21) for changes in <cond> with the default processor setting.

Example:

```
C>TI EVX1,2/234,3 ← This command will measure
the time from the second positive transition of
EXTERNAL EVENT 1 (high on 1, don't care on 2) to
the third occurrence of address 234 after the EXTER-
NAL EVENT condition has been met.
```

TIME ENABLED

```
EVX1 OCCUR:2 TO A:234 OCCUR:3 GO:N
```

```
C>TI 350, 1/24,2,G ← This command will measure
the elapsed time from the first occurrence of address
350 to the second occurrence of address 24. It will
repeat this until interrupted from the keyboard.
```

```
A:350 OCCUR:1 TO A:024 OCCUR:2 GO:Y
```

```
C>TI 44
```

TIME ENABLED

```
A:044 OCCUR:1 TO A:044 OCCUR:1 GO:N
```

```
C>G ← This example shows the default conditions
of the command. Used with the previous example,
this command will measure the elapsed time
between the first occurrence of address 44 and the
next occurrence of address 44.
```

### 7.3.25 TYPE Command

Syntax: TYPE [<print opt>],[<print opt>]. . .]

Prints out the information specified to the printer or console. As with the AUTOPRINT command, if a RAM register is specified, its 16-digit contents will be listed, from left to right, most-significant digit to least-significant digit. If no options are specified and a trace operation was just executed, trace memory will be displayed in blocks of 16. When printing trace memory while the chip is breakpointed, the last eight locations of trace memory will not be displayed.

Example:

```
C>T P, Q, B, M1F M2
```

```
B:10 Q:FF P:004 OF LBI 0 M1F:0 M2:0000000120F120E
```

COP2440, COP2441, COP2442 users: See 'SET PROC-MODE' command (Section 7.3.21) for changes in <print opt> with the default processor setting.

### 7.3.26 TRACE Command

Syntax: TRACE [<cond>],[<occur#>],[<prior> [<gopt>]]]

Sets the print trace conditions. During a TRACE operation, COPMON stores each consecutive value of the COP program counter in a 254-word circular buffer, so that at any time during trace operation, the buffer has the previous 254 values of the program counter. When <cond> has been met, the number of times specified by <occur#>, COPMON saves the number of values of the program prior to <cond> specified by <prior>, and fills the rest of the buffer with the subsequent values of the program counter. It then prints the <cond> specified and the address where <cond> was recognized, followed by any trace data specified by the AUTOPRINT command. If <cond>,< occur#> or <prior> are omitted, they retain their previous values. If <gopt> is included, then each time a trace operation is finished, another GO command is performed with the same conditions, continuing until interrupted by the console. The TRACE command does not initiate trace operation, but sets the Trace Enable flag so that trace operation is initiated on the next GO command. See Table 7-4.

Example:

```
C>TR EV0X, 2, 22
```

TRACE ENABLED:

```
EV0X OCCUR:2 PRIOR:22 G:N
```

Under certain conditions (see Table 7-4), the <prior> count specified may not be fulfilled. That is, <cond> may occur before <prior> cycles of the chip. In this case, when typing trace memory, a message of the form "ONLY nn PRIOR LOCATIONS TRACED" will appear.

Example: Assume that all of shared memory contains NOP instructions, except location 0, which has a CLRA instruction.

```
C>R
```

CHIP IS RESET

```
C>AU A,P
```

```
C>S
```

STEP

```
A:0 P:001
```

```
C>TR 5,1,245
```

TRACE ENABLED

```
A:005 OCCUR:1 PRIOR:245 GO:N
```

```
C>G
```

TRACED ON A:005 AT A:005

```
C>T 0/250
```

ONLY 4 PRIOR LOCATIONS TRACED

Table 7-2. Summary of COPMON Console Commands

Command Name	Operand Syntax	Description
ALTER	A[<addr>][,<value>] . . .	Alter Shared Memory
AUTOPRINT	AU[<print opt>[,<print opt>] . . .]	Set Print Options
BREAKPOINT	B[<cond>[/<cond>] . . .][,<occur#>[,<gopt>]]	Set Breakpoint
CLEAR	C	Clear Trace and Breakpoint Flags
CHIP	CH <chip#>	Set or Display Chip Number
COMPARE	CO <filename>	Compare File with Shared Memory
DEPOSIT	D <value>,<addr range>	Deposit Values into Shared Memory
FIND	F <value>[,<addr range>[,<mask>]]	Find Value in Shared Memory
GO	G[<addr>]	Begin Program Execution
GO	G[<addrx>][,<addy>]	(Dual Processor Chips Only)
HELP	H	Display Command Summary
LIST	L[<addr range>[,<addr range>] . . .]	List Shared Memory
LOAD	LO<filename> [O]	Load Shared Memory from File
MODIFY	M[<print opt>,<value>[,<value1>] . . .]	Modify Registers and COPS RAM
NEXT	N[<gopt>]	Breakpoint on Next Instruction
NEXT	N[<gopt>]X[,<gopt>]Y[,<gopt>]	(Dual Processor Chips Only)
PUT	P[<addr>][,<instruct>[,<instruct>] . . .]	Put Instruction (Assemble)
RESET	R	Reset Chip
SINGLESTEP	S[<gopt>]	Single-Step
SINGLESTEP	S[<gopt>]X[,<gopt>]Y[,<gopt>]	(Dual Processor Chips Only)
SAVE	SA<filename>	Save Shared Memory into File
SEARCH	SE<addr>	Search for Address in Trace Memory
SET	SET SI {Y N} or SET ST {Y N}	Set SIOMODE or STACKMODE Flags
SET	SET PR {X Y D}	Set Default Processor Mode. Dual Processor Only.
SHARED MEM	SH {Y N}	Set/Clear Shared Memory Mode
STATUS	ST	Display Chip Status
TIME	TI[<cond1>[,<occur1>]I[<cond2>[,<occur2>] [,<gopt>]]]	Measure Elapsed Time
TYPE	T[<print opt>[,<print opt>] . . .]	Type Breakpoint or Trace Data
TRACE	TR[<cond>[,<occur#>[,<prior>[,<gopt>]]]]	Set Trace Conditions
UNASSEMBLE	U{Y N}	Display Instruction Mnemonics of the Data in Shared Memory

Table 7-3. Operand Syntax

Operand	Description
<addr>	One to three hexadecimal digits, < = maximum address of the chip defined by <chip#>. P = Previous address. . = Current address, i.e., last address altered or typed. N = Next address. L = Last address defined by chip number.
<addr cond>	Address in hexadecimal, greater than 0, less than or equal to maximum address of chip.
<addr range>	<addr>[/<addr>].
<chip#>	410, 411, 420, 421, 422, 444, 445, 440, 441, 442, 2440, 2441 or 2442.
<cond>	<addr cond> <evt cond>
<dig#>	Hexadecimal digit specifying RAM digit address (see Table 7-1).
<end>	Decimal 0-253; last location of trace memory desired. (See Note 1.)
<evt cond>	EV00, EV01, EV10, EV11, EVX0, EVX1, EV0X, or EV1X. Format: EV<EXT2><EXT1>. "1" = Logic 1 "0" = Logic 0 "X" = Don't Care
<filename>	Valid PDS filename, default extension assumed is .LM.
<gopt>	G = Go immediately after printing.
<instruct>	Valid COPS instruction mnemonic with operand. The operand is hexadecimal.
<mask>	Hexadecimal 0-0FFH (0-0FFFFH for a 2-byte FIND).
<occur#>	Decimal 1-256. Number of times <cond> occurs before initiating Breakpoint, Trace, or Time.
<print opt> (See Note 2.)	A = Accumulator (BR,M). ALL = All Breakpoint Data (BR). B = RAM Address Register B (BR,M). C = Carry Bit (BR,M). D = Output Port (M). E = EN Register (M). (See Note 3.) G = G I/O Port (BR,M). H = H I/O Register (BR,M). I = I Input Port (BR). L = L I/O Port (BR). M = All RAM on Chip (BR). M<reg#> = RAM Register <reg#> (BR,M). M<reg#><dig#> = RAM Digit <reg#><dig#> (BR,M). N = N (stack pointer) Register (BR,M). P = Program Counter (BR). R = R I/O Register (BR,M). S = Serial I/O Register (only if SIOMODE is true) (BR,M). SA = Stack Register SA. ----: SB = Stack Register SB. :- (BR,M) SC = Stack Register SC. : (See Note 4.) SD = Stack Register SD. ----: ST = All Stack Registers (only if STACKMODE is true) (BR). T = Trace Memory 0 Through 253 (BR,TR). (See Note 2.) TI = T (Timer) Register (BR,M). <start> = Trace Memory Location<start> (BR,TR). <start>/<end> = Trace Memory <start> Through <end> (BR,TR). AX,BX,CX,NX = A,B,C & N Registers of Processor X (BR,M). AY,BY,CY,NY = A,B,C & N Registers of Processor Y (BR,M). PX = Program Counter, Processor X (BR). PY = Program Counter, Processor Y (BR). SAX,SBX,SCX,SDX = Stack Registers of Processor X (BR,M). SAY,SBY,SCY,SDY = Stack Registers of Processor Y (BR,M). STX = All Stack Registers of Processor X (BR). STY = All Stack Registers of Processor Y (BR).

**Table 7-3. Operand Syntax (Continued)**

Operand	Description
<prior>	Decimal 0-253; Number of Addresses Traced Prior to <cond>. (See Note 2.)
<proc>	X Y D Designates Processor X, Y, or Dual.
<reg#>	Hexadecimal Digit Specifying RAM Register.
<start>	Decimal 0-253; First Location in Trace Memory Desired. (See Note 2.)
<value>	Hexadecimal 0-0FFH.

**Note 1:** If using a Dual processor COPS in Dual mode, the maximum value is limited to 252.

**Note 2:** Print Options listed with (BR) apply to breakpoint and single-step operations, those listed with (TR) apply to trace operations, and those listed with (M) apply to the MODIFY command.

**Note 3:** Also applies to breakpoint and single-step (BR) for COP440, COP441, COP442, COP2440, COP2441 and COP2442.

**Note 4:** Valid only if STACKMODE is true. Also, for COP440, COP441, COP442, COP2440, COP2441 and COP2442, if not a valid stack entry as indicated by the stack pointer reg N, a '?' is printed after the entry. For example, on the COP440, if N = 1, SA and SB are printed as SA:023 SB:344.

**Table 7-4. GO Operation Summary**

Address Given	BRKPT or TRACE Enabled	COP Status	Function Performed
No	No	Reset	Start chip at addr 000.
No	No	Breakpointed	Start chip at BRK addr.
No	No	Running	"COP ALREADY RUNNING."
No	BRKPT	Reset	Start chip at addr 000, enter breakpoint mode.
No	BRKPT	Breakpointed	Start chip at BRK addr, enter breakpoint mode.
No	BRKPT	Running	Enter breakpoint mode.
No	TRACE	Reset	Start chip at addr 000, enter trace mode.
No	TRACE	Breakpointed	This is allowed, but the prior count condition specified by the user in enabling TRACE may not be fulfilled.
No	TRACE	Running	Enter TRACE mode.
Yes	No	Reset	Breakpoint at 1, start chip at (ADDR).
Yes	No	Breakpointed	Start chip at (ADDR).
Yes	No	Running	"COP ALREADY RUNNING."
Yes	BRKPT	Reset	Breakpoint at 1, start chip at <addr>, enter breakpoint mode.
Yes	BRKPT	Breakpointed	Start chip at <addr>, enter breakpoint mode.
Yes	BRKPT	Running	Breakpoint immediate, start chip at <addr>, enter breakpoint mode.
Yes	TRACE	Reset	Breakpoint at 1, start chip at <addr>, enter trace mode.
Yes	TRACE	Breakpointed	This is allowed, but the prior count condition specified by the user in enabling TRACE may not be fulfilled.
Yes	TRACE	Running	Breakpoint immediately, start chip at <addr>, enter trace mode.

**Note:** The function of the GO command depends on the mode that the COPS chip is in whether or not BRKPT or TRACE or TIME is enabled, and whether or not <addr> is given.

The TIME enable flag has the same effects as the TRACE flag, i.e., if TIME is enabled, just substitute TIME for TRACE in the table.

```

241 -4 A:001 E:1111
242 -3 A:002 E:1111
243 -2 A:003 E:1111
244 -1 A:004 E:1111
245 0 A:005 E:1111
246 1 A:006 E:1111
247 2 A:007 E:1111
248 3 A:008 E:1111
249 4 A:009 E:1111
250 5 A:00A E:1111
    
```

External Event lines, EXT4 to EXT1.  
 Program Counter value.  
 Location relative to <cond> (trace condition).  
 Location in Trace Buffer.

COP2440, COP2441 and COP2442 users: See 'SET PROCMODE' command for changes in <cond> with the default processor setting.

Also, when in DUAL mode, both processors are traced, and trace memory is restricted to locations 0 through 252. Processor X is displayed on the left-hand side of the screen, processor Y on the right-hand side.

If the mode is X-only or Y-only, only that processor is traced.

**7.3.27 UNASSEMBLE Command**

Syntax: UNASSEMBLE {Y|N}

Gives an opcode and mnemonic for each instruction. This command selects the unassemble mode for use during trace and list operations. If a LIST is started on the second byte of a 2-byte instruction, the unassembly will be incorrect until two successive 1-byte instructions are encountered.

# File List Program (LIST)

## 8.1 Introduction

The File List program (LIST) provides the user with a means of listing any type of file on the system console or printer. LIST has several print options available that allow setting of page headings, control of page and line numbers, etc. Files that are not symbolic are listed in hexadecimal and ASCII.

## 8.2 Invoking LIST

To call LIST, the user types in the @ command:

```
X>@LIST<filename>[<options>]
```

```
LIST,REV:A
```

(Listing now begins.)

or

```
X>@LIST
```

```
LIST,REV:A
```

```
L><filename>[<options>]
```

(Listing now begins.)

where filename is the name of the file to be listed, and options are print options described below. Table 8-1 summarizes all LIST options. Each may be abbreviated to the first two characters. The default modifier for the filename is SRC.

Upon completion of all copies of the listing, LIST prompts the user for another filename and options.

## 8.3 Options

Options are scanned left to right and are separated from the @LIST command and other options by spaces. The order is significant to the extent that later options may change those previously specified. Otherwise, the only other requirement is that the HD option must be last (since it is followed by text rather than more options). All option names can be abbreviated to the first two characters. In the option parameters, n is a decimal number and char is a single ASCII character.

### 8.3.1 Device Output Options

Device Selection Options specify the output device, line width, and spacing between pages.

Option	Description
(default)	Six line feeds between page on output listing.
NL n	Set number of nulls to be output after a carriage return; used for generating paper tapes to be read on other systems.
P or PR	Select Printer output and generate form feed between pages.
TTY	TTY new page: send form feeds instead of line feeds.
WAIT	TTY new page: wait for input character to resume.

Table 8-1. Summary of LIST Options

Option	Meaning
ASNI	File contains ANSI carriage-control character in column 1.
CO n	Select number of copies to print.
DBL	Double space.
FO char	Set form feed character.
HD text	Set heading text.
HEX	Print file in unformatted hex/ASCII.
IN n	Indent left margin n spaces.
LI n/n	Select line range to print.
LP n/n	Set number of lines per page.
NE	Suppress page eject after 58 lines.
NF	Suppress formatting (same as combination of NE, NH, NP and UN).
NH	Set number of nulls following a carriage return.
NL n	Suppress page eject when encountering assembler directive .FORM.
NP	Print line numbers on listing.
N or NU	Select page range to print.
PA n/n	Eject page when encountering assembler directive .FORM.
P or PR	Select printer output.
QU	Compress blanks in the output (quick mode).
TAB	Print the tab character.
TR n	Set right margin at n characters and truncate characters exceeding the margin.
TTY	Send form feed on new pages.
UN	Suppress line numbers on listing.
WAIT	Wait for any input character before resuming listing on new pages.
WI n	Set right margin at n characters and start new line for characters exceeding the margin.

### 8.3.2 Text Formatting Options

Text Formatting Options control the actual text that is printed.

Option	Description
ANSI	Treat the first character of the text line as an ANSI carriage-control character. The control character is not printed but controls the line spacing instead. The control characters are: + = No spacing (overprint) - = Triple space 0 = Double space 1 = New page Anything else = Single space
DBL	Double space. If used together with the ANSI option, the specified spacing is doubled.
FO char	Set form feed character. If this character occurs in column 1 of the text line, a new page occurs (instead of printing the character).
HEX	Print file in unformatted hexadecimal/ASCII. Other formatting options will not apply. This option is always taken for files other than Symbolic or Data files.
N or NU	Print line numbers on listing (default for files with modifier of ".SRC").
QU	Quick Mode: compress blanks in the printout. Any sequence of consecutive blanks in a line image is printed as a single blank. This option will negate any indent that may be set.

TAB	Print the tab character (09). Otherwise the tab is printed as the number of spaces required to move to the next fixed tab stop (every eight columns in the text field).
UN	Do not print line numbers (default for all files except for those with a modifier of ".SRC").

### 8.3.3 Line Control Options

Line Size Options control indentation and margin.

Option	Description
IN n	Indent left margin n spaces (default = 0).
TR n	Set right margin at n characters; any additional characters are truncated and not printed.
WI n	Set right margin at n characters; any additional characters are folded over onto a new line (default = 72).

Note: the PR Option sets the width to 80, but does not change the fold/truncate mode.

### 8.3.4 Printing Options

Print Selection Options determine what part of the data is to be printed.

Option	Description
CO n	Select number of copies to print.
LI n/n	Select line range to print. May be specified as n or n/n; the first number is the first line to be printed and the second number is the last line to be printed.
PA n/n	Select page range to print; action is the same as LI option except that the range is by page number.

Both LI and PA may be specified; the action is as follows: If either range starts at 1, the starting number of the other option determines the first line to be listed. The first end specification encountered stops the listing. If a single line or page number is specified, printing begins with that line or page and continues to the end of the file.

### 8.3.5 Page Control Options

Page/Heading Options control the number of lines on a page, page heading, and page printing.

Option	Description
(default)	Lines are counted so that a new page occurs after every 58 text lines, thus providing proper formatting on 8½ × 11" pages. If the file modifier is .LST, the NF option is assumed. If the file modifier is SRC, the PG option is assumed; otherwise NP is assumed.
HD text	Set heading text. The default is the name of the file, e.g., LIST.SRC. This must be the last option on the calling line since all characters following the HD up to the carriage return are used for the heading text.
LP n/n	Set number of lines per page. The first number is the number of text lines on a page; the second number is the number of lines on a sheet of paper. The default is 58/66, which is correct for 8½ × 11" pages. It is not necessary to specify both numbers if only the number of text lines is to be changed. The text will be automatically centered vertically on the page.
NE	Suppress page eject on counted line; page ejects occur only where explicitly determined by the text (either form feeds or assembler directive .FORM, if being checked).
NF	No formatting (same as NE, NH, NP, and UN). This option is the default for files with a modifier of ".LST".
NH	Suppress heading at top of page.
NP	Suppress page eject when encountering assembler directive .FORM
PG	Check for assembler directive .FORM. This option is the default for files with a modifier of ".SRC".

Example: List on printer ADD.SRC file, no formatting except for line number printout:

```
X> @LIST ADD.SRC PR NE NH NP
```



# Cross Reference Program (XREF)

## 9.1 Introduction

The Cross Reference program (XREF) is a PDS system program that provides a means for printing a symbol map of COP assembly language programs. The symbol map shows the name of every symbol in the program, the line number where the symbol is defined, and all of the line numbers where the symbol is used.

## 9.2 Invoking XREF

To call XREF, the user types in the @ command:

```
X>@XREF<filename>
XREF,REV:A
(Cross referencing now begins.)
or
X<@XREF
XREF,REV:A
R><filename>
(Cross referencing now begins.)
```

where <filename> is the name of the COP assembly language (SYMBOLIC) file to be cross referenced. If the filename is followed by "\*\*PR", the listing will be printed on the printer.

Figure 9-1 shows a typical cross reference listing. Local symbols (i.e., those starting with a \$ sign) are listed first. Symbols are listed in alphabetical order. The numbers listed beside each symbol are the numbers of the lines in which the symbol appears. A "-" beside a line number indicates that the symbol is assigned or otherwise given a value in that line. A "\*" beside a symbol means that the symbol appears in only one line.

```
X>@XREF PDS:COPPGM.SRC
FILE PDS:COPPGM.SRC
```

\$5	653	703-								
\$8	538	557	560-							
\$DF1	258	287-								
\$DF2	260	88-								
\$END	536	563-								
\$LOOP	533-	552								
\$SAVE	418	437	450-	577	592	595-				
A	71-	103	104	112	113	133		134		
ADD1	88	326-								
ADDSYM	338	392-								
AFST	47-	257	525							
B	103	105	112	113	783-	789	790	791	792	793
	795	796	797	798	799	801	803	804	805	809
	820	821	822	824	825	829	830	832	835	837
	843	844	846	847	851	852	854	855	857	861
	863	864	867	872	874					
BEG	154-	170	171	172	173	190	200	204	205	205
	206	206	207	208						
IQT	157-	169	189	189	190	191	191	192	192	193
	193	194	194							
JTAB	81-	312								
L	66-	136	137	137	137	137	138	138	138	138
	139	139	139	139	140	140	140	140	141	141
	144	145	145	145	145	146	146	146	146	147
	147	147	147	148	148	148	148	149	149	
LAST	28-	237	336	391	400	420	652	722		
LBYT	468	472-								
LINCNT	41-	648	708	713						
LINE	39-	233	432	529	532					
MAIN	269	296-	313							
MAXRAM	60-	238								
MESG	51-	253	271	396	446					
MTOP	61-	239								
N	68-	131	131	131	131	132	132	132	133	
	133									
SYM	155-	168	170	171	173	175	175	176	176	180
	182	183	187	203						
T	77-	127								
TEMP	43-	368	369	539	545	673	690			
TOVFLW	397	447	451-							
X	67-	141	149							
*XREF	1									
XX	104	107								
YY	105	107								
ZRO	19-	466	583	685						
XX	106	107								

Figure 9-1. Typical Cross Reference Listing

# Mask Transmittal Program (MASKTR)

## 10.1 Use of Mask Transmittal Program

MASKTR is a PDS system program which is used to generate a transmittal file that NSC uses for creating the COP chip ROM/OPTIONS mask and the functional test type is SYT.

The transmittal filename will be the same as the LOAD Module filename, the modifier will be .TRN, and the internal file type is SYT.

The transmittal file contains:

1. Name and phone number of the responsible person.
2. Company name and address.
3. Date.
4. Chip Number.
5. Listing of options showing option number, option name and option value.
6. ROM data including addresses. Unused addresses are set to OP CODE zero (0), which is a CLRA instruction.
7. Source, object, and transmittal file checksums.

To enter any information for the TRANSMITTAL file, MASKTR must first be in the TRANSMITTAL command (T) or by typing the filename on the end of the '@MASKTR' line.

When MASKTR is in the TRANSMITTAL mode, the user is requested to provide the following information:

1. LOAD MODULE FILENAME.
2. CHIP NUMBER.
3. NAME AND PHONE NUMBER OF RESPONSIBLE PERSON.
4. COMPANY NAME AND ADDRESS.
5. DATE.
6. OPTION VALUES.

MASKTR prompts the user with a description of the desired item required by the program, the current value of the data-item (as last entered by the user), and then asks for the new value from the user. If no change is required, a carriage return will leave the value unchanged; if a change is requested for the chip number or options, the value entered is checked for validity. Entering a blank line causes an advance to the next item to be entered.

The items are arranged in a circular order such that the user will be prompted for responsible person (name/phone), company (name/address), date, chip number, options, and then back to responsible person in that order.

Note: A CNTL/Q in column 1 causes a return to the prompt mode.

To call MASKTR, type:

```
X>@MASKTR
  MASKTR, REV:C, DATE
  T>
```

MASKTR is then ready to accept one of the commands listed in Table 10-1 and described hereafter.

## 10.2 ABORT Command

Syntax: ABORT

Aborts the creation of a transmittal file and returns the program to the PROMPT mode.

Example:

```
T>A
  ABORT TRANSMITTAL FILE CREATION
  (Y/N, CR = YES)? CR
  TRANSMITTAL FILE CREATION ABORTED
  T>
```

## 10.3 CHIP Command

Syntax: CHIP

Prompts the user for the chip number.

Example:

```
T>C
  CHIP NUMBER: 420L
  CHIP NUMBER: 320L
  EXTENDED TEMPERATURE RANGE
  (Y/N, CR = YES)? CR
```

Note: The chip number must be specified in the above manner if parts with extended temperature range are desired.

## 10.4 COMPANY Command

Syntax: COMPANY

Prompts the user for the company name and address. Eight lines are allowed for this entry.

Example:

```
T>C
  COMPANY NAME AND ADDRESS:
  UNSPECIFIED
  COMPANY NAME AND ADDRESS:
  NATIONAL SEMICONDUCTOR
  2900 SEMICONDUCTOR DRIVE
  SANTA CLARA, CA 95051
  CR
  DATE: UNSPECIFIED
```

## 10.5 DATE Command

Syntax: DATE

Prompts the user for the date. One line is allowed for this entry,

Example:

```
T>D
  DATE: UNSPECIFIED
  DATE: 1 JANUARY, 1979
  CHIP NUMBER: 420
```

## 10.6 FINISH Command

Syntax: FINISH

Finishes the creation of the transmittal file, and writes it to the disk. There is a prompt for the disk to be sent to NSC to be placed in the drive. Note: The disk must be an initialized disk.

Example:

```
T> F
FINISH CREATION OF TRANSMITTAL FILE
(Y/N,CR = YES)? CR
INCOMPLETE OPTION SPECIFICATION
T>
```

Note: The user must completely define all options before the program will allow a transmittal file to be written to the disk.

The FINISH command also checks for conflicting CKO-CKI option selections and option selections which are illegal for a bonding option value of 2.

## HELP Command

Syntax: HELP

Lists command summary.

## 10.8 LIST Command

Syntax: LIST

Lists the transmittal file as it will appear on the form that will be returned to the customer from NSC for verification and sign-off before a mask will be generated from the customer's transmittal disk.

Note: A \*PR at the end of this command will cause the listing to go to the printer.

Example:

```
T> L
```

This example will list the transmittal file on the console in blocks that will fit on the screen. A CR will advance to the next block of data. Any other key followed by a CR will return to the PROMPT. A CNTL/Q will also return to the PROMPT mode.

## 10.9 NAME Command

Syntax: NAME

Prompts the user for the name/phone number of the person responsible for this program. Two lines are allowed for this entry.

Example:

```
T> N
RESPONSIBLE NAME/PHONE:
UNSPECIFIED
RESPONSIBLE NAME/PHONE:
JOE USER
123 456 7890
COMPANY NAME/ADDRESS:
```

## 10.10 OPTION Command

Syntax: {OPTION [<opt#>]<opt#>}

Prompts the user for the valid options for the chip specified. If the "0" is omitted the <opt#> must be specified. If the "0" is inserted and <opt#> is omitted, the program prompts for options from the first option.

Example:

```
T> O 12
OPTION 12: L3 DRIVER = UNSPECIFIED
00 = STANDARD OUTPUT
01 = OPEN DRAIN
02 = HI CURRENT LED SEG OUT
03 = HI CURRENT TRI-STATE
04 = LOW CURRENT LED SEG OUT
05 = LOW CURRENT TRI-STATE
OPTION 12: L3 DRIVER O1
OPTION 12: L3 DRIVER = 01 (Y/N, CR = YES)? CR
OPTION 13: L2 DRIVER = UNSPECIFIED
```

## 10.11 PRINT Command

Syntax: PRINT

Prints out the allowable options for the chip specified in the command.

Note: If a \*PR is entered at the end of the line, the options are sent to the printer.

Example:

```
T> P 420
ABORT TRANSMITTAL FILE CREATION
(Y/N, CR = YES)? CR
TRANSMITTAL FILE CREATION ABORTED
T> P 420
CHIP NUMBER: 420
OPTION 1: GROUND
NOT AN OPTION
OPTION: CKO OUTPUT
00 = CLOCK GEN OUT XTAL/RES
01 = RAM KEEP ALIVE
02 = GENERAL INPUT, VCC LOAD
03 = MULTICOP SYNC IN
04 = GENERAL INPUT, HI-Z
```

This example will print the COP420 options on the console. As in the LIST command, the block of data is sent to the screen and a CR will advance through the options. A \*PR will send the options to the printer. The print command cannot be used while in the TRANSMITTAL mode.

## 10.12 TRANSMITTAL Command

Syntax: TRANSMITTAL <filename>

When the TRANSMITTAL command is invoked, the chip number prompt is given. The LOAD MODULE is read into memory, and the entered chip number is checked against the chip number contained in the LOAD MODULE (assembled with REV B ASM). If the chip numbers do not match, the program aborts the

TRANSMITTAL command and returns to Prompt. If the chip numbers agree, the valid chip number is entered into the data table and used to determine which options are valid and available. The ROM data and option values (if any) from the LOAD MODULE are also entered into the data table.

The TRANSMITTAL command may also be invoked by typing the filename on the @MASKTR line.

#### Example

```
EXEC, REV:A
X>@MASKTR
MASKTR,REV:B, DATE
M>T MASKEK
DISK WITH LOAD MODULE IN DRIVE
(Y/N, CR = YES)? CR
CHIP NUMBER: UNSPECIFIED
CHIP NUMBER: 421
CHIP NUMBER: 421
CHIP NUMBER: CR
ERROR: PROGRAM ASSEMBLED FOR 420
M>T MASKEK
DISK WITH LOAD MODULE IN DRIVE
(Y/N, CR = YES)? CR
CHIP NUMBER: UNSPECIFIED
CHIP NUMBER: 420
CHIP NUMBER: 420
CHIP NUMBER: CR
RESPONSIBLE NAME/PHONE:
UNSPECIFIED
RESPONSIBLE NAME/PHONE:
JOE COPUSER
(415) 777-6234
COMPANY NAME/ADDRESS:
UNSPECIFIED
COMPANY NAME/ADDRESS:
NATIONAL SEMICONDUCTOR
2900 SEMICONDUCTOR DRIVE
SANTA CLARA, CA 95051
CR
DATE: UNSPECIFIED
DATE: JANUARY 5, 1979
CHIP NUMBER: 420
CHIP NUMBER: CR
OPTION 01: GROUND = 00
NOT AN OPTION
OPTION 02: CKO OUTPUT = 02
00 = CLOCK GEN OUT XTAL/RES
01 = RAM KEEP ALIVE
02 = GENERAL INPUT, VCC LOAD
03 = MULTICOP SYNC IN
04 = GENERAL INPUT, HI-Z
OPTION 02: CKO OUTPUT CR
OPTION 03: CKI INPUT = 04
00 = XTAL/16
01 = XTAL/8
02 = TTL/16
03 = TTL/8
04 = RC/4
05 = OSC = (SCHMITT IN)/4
```

```
OPTION 03: CKI INPUT CR
OPTION 04: RESET INPUT = 00
00 = LOAD VCC
01 = HI-Z
OPTION 04: RESET INPUT 1
OPTION 04: RESET INPUT = 01
(Y/N, CR = YES)? CR
OPTION 05: L7 DRIVER = 02
00 = STANDARD OUTPUT
01 = OPEN DRAIN
02 = HI CURRENT LED SEG OUT
03 = HI CURRENT TRI-STATE
OPTION 05: L7 DRIVER CR
OPTION 06: L6 DRIVER = 02
00 = STANDARD OUTPUT
01 = OPEN DRAIN
02 = HI CURRENT LED SEG OUT
03 = HI CURRENT TRI-STATE
OPTION 06: L6 DRIVER CR
OPTION 07: L5 DRIVER = 02
00 = STANDARD OUTPUT
01 = OPEN DRAIN
02 = HI CURRENT LED SEG OUT
03 = HI CURRENT TRI-STATE
OPTION 07: L5 DRIVER CR
OPTION 08: L4 DRIVER = 02
00 = STANDARD OUTPUT
01 = OPEN DRAIN
02 = HI CURRENT LED SEG OUT
03 = HI CURRENT TRI-STATE
OPTION 08: L4 DRIVER CR
OPTION 09: IN 1 INPUT = 00
00 = TTL LOAD
01 = TTL HI-Z
OPTION 09: IN 1 INPUT CR
OPTION 10: IN 2 INPUT = 00
00 = TTL LOAD
01 = TTL HI-Z
OPTION 10: IN 2 INPUT CR
OPTION 11: VCC = 00
NOT AN OPTION
OPTION 12: L3 DRIVER = 02
00 = STANDARD OUTPUT
01 = OPEN DRAIN
02 = HI CURRENT LED SEG OUT
03 = HI CURRENT TRI-STATE
OPTION 12: L3 DRIVER CR
OPTION 13: L2 DRIVER = 02
00 = STANDARD OUTPUT
01 = OPEN DRAIN
02 = HI CURRENT LED SEG OUT
03 = HI CURRENT TRI-STATE
OPTION 13: L2 DRIVER CR
```

- OPTION 14: L1 DRIVER = 02  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN  
 02 = HI CURRENT LED SEG OUT  
 03 = HI CURRENT TRI-STATE
- OPTION 14: L1 DRIVER CR
- OPTION 15: L0 DRIVER = 02  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN  
 02 = HI CURRENT LED SEG OUT  
 03 = HI CURRENT TRI-STATE
- OPTION 15: L0 DRIVER CR
- OPTION 16: SI INPUT = 00  
 00 = LOAD VCC  
 01 = HI-Z
- OPTION 16: SI INPUT CR
- OPTION 17: SO DRIVER = 02  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN  
 02 = PUSH/PULL
- OPTION 17: SO DRIVER CR
- OPTION 18: SK DRIVER = 02  
 00 = STANDARD OUTPUT  
 02 = OPEN DRAIN  
 03 = PUSH/PULL
- OPTION 18: SK DRIVER CR
- OPTION 19: IN 0 INPUT = 00  
 00 = TTL LOAD  
 01 = TTL HI-Z
- OPTION 19: IN 0 INPUT CR
- OPTION 20: IN 3 INPUT = 00  
 00 = TTL LOAD  
 01 = TTL HI-Z
- OPTION 20: IN 3 INPUT CR
- OPTION 21: G0 I/O PORT = 00  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN  
 02 = STANDARD OUTPUT SMALL DRIVER  
 03 = OPEN DRAIN SMALL DRIVER
- OPTION 21: G0 I/O PORT CR
- OPTION 22: G1 I/O PORT = 00  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN  
 02 = STANDARD OUTPUT SMALL DRIVER  
 03 = OPEN DRAIN SMALL DRIVER
- OPTION 22: G1 I/O PORT CR
- OPTION 23: G2 I/O PORT = 00  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN  
 02 = STANDARD OUTPUT SMALL DRIVER  
 03 = OPEN DRAIN SMALL DRIVER
- OPTION 23: G2 I/O PORT CR
- OPTION 24: G3 I/O PORT = 00  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN  
 02 = STANDARD OUTPUT SMALL DRIVER  
 03 = OPEN DRAIN SMALL DRIVER
- OPTION 24: G3 I/O PORT CR
- OPTION 25: D3 OUTPUT = 00  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN
- OPTION 25: D3 OUTPUT CR
- OPTION 26: D2 OUTPUT = 00  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN
- OPTION 26: D2 OUTPUT CR
- OPTION 27: D1 OUTPUT = 00  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN
- OPTION 27: D1 OUTPUT CR
- OPTION 28: D0 OUTPUT = 00  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN
- OPTION 28: D0 OUTPUT CR
- OPTION 29: COP FUNCTION = 00  
 00 = NORMAL  
 01 = MICROBUS
- OPTION 29: COP FUNCTION CR
- OPTION 30: COP BONDING = UNSPECIFIED  
 00 = 28 PIN PACKAGE  
 01 = 24 AND 28 PIN PACKAGES
- OPTION 30: COP BONDING 2
- OPTION 30: COP BONDING = 02  
 (Y/N, CR = YES)? CR
- OPTION 31: IN INPUT LEVEL CR  
 00 = STANDARD OUTPUT  
 01 = HIGH TRIP POINT
- OPTION 31: IN INPUT LEVEL CR
- OPTION 32: G INPUT LEVEL = UNSPECIFIED  
 00 = STANDARD OUTPUT  
 01 = HIGH TRIP POINT
- OPTION 32: G INPUT LEVEL 1
- OPTION 32: G INPUT LEVEL = 01  
 (Y/N, CR = YES)? CR
- OPTION 33: L INPUT LEVEL = UNSPECIFIED  
 00 = STANDARD OUTPUT  
 01 = HIGH TRIP POINT
- OPTION 33: L INPUT LEVEL 1
- OPTION 33: L INPUT LEVEL = 01  
 (Y/N, CR = YES)? CR
- OPTION 34: CKO INPUT LEVEL = UNSPECIFIED  
 00 = STANDARD TTL  
 01 = HIGH TRIP POINT
- OPTION 34: CKO INPUT LEVEL 0
- OPTION 34: CKO INPUT LEVEL = 00  
 (Y/N, CR = YES)? CR

OPTION 35: SI INPUT LEVEL = UNSPECIFIED

00 = STANDARD TTL  
01 = HIGH TRIP POINT

OPTION 35: SI INPUT LEVEL 0

OPTION 35: SI INPUT LEVEL = 00  
(Y/N, CR = YES)? CR

RESPONSIBLE NAME/PHONE:

JOE COPUSER  
(415)777-6234

RESPONSIBLE NAME/PHONE: CR

COMPANY NAME/ADDRESS:

NATIONAL SEMICONDUCTOR  
2900 SEMICONDUCTOR DRIVE  
SANTA CLARA, CA 95051  
USA

COMPANY NAME/ADDRESS: CNTL/Q

#

M>L

PDS TRANSMITTAL FILE

RESPONSIBLE NAME/PHONE:

JOE COPUSER  
(415) 777-6234

COMPANY NAME/ADDRESS:

NATIONAL SEMICONDUCTOR  
2900 SEMICONDUCTOR DRIVE  
SANTA CLARA, CA 95051  
USA

DATE: JANUARY 5, 1979

FILE NUMBER: B8A7 62A0 102B

CHIP NUMBER: 420

OPTION	VALUE	OPTION	VALUE
01: GROUND	= 00	19: IN 0 INPUT	= 00
02: CKO OUTPUT	= 02	20: IN 3 INPUT	= 00
03: CKI INPUT	= 04	21: G0 I/O PORT	= 00
04: RESET INPUT	= 01	22: G1 I/O PORT	= 00
05: L7 DRIVER	= 02	23: G2 I/O PORT	= 00
06: L6 DRIVER	= 02	24: G3 I/O PORT	= 00
07: L5 DRIVER	= 02	25: D3 OUTPUT	= 00
08: L4 DRIVER	= 02	26: D2 OUTPUT	= 00
09: IN 1 INPUT	= 00	27: D1 OUTPUT	= 00
10: IN 2 INPUT	= 00	28: D0 OUTPUT	= 00
11: VCC	= 00	29: COP FUNCTION	= 00
12: L3 DRIVER	= 02	30: COP BONDING	= 02
13: L2 DRIVER	= 02	31: IN INPUT LEVEL	= 00
14: L1 DRIVER	= 02	32: G INPUT LEVEL	= 01
15: L0 DRIVER	= 02	33: L INPUT LEVEL	= 01
16: SI INPUT	= 00	34: CKO INPUT LEVEL	= 00
17: SO DRIVER	= 02	35: SI INPUT LEVEL	= 00
18: SK DRIVER	= 02		

Table 10-1. MASKTR Command Summary

Command	Syntax	Description	Section
ABORT	A	Abort transmittal file creation.	10.2
CHIP	C	Enter chip number.	10.3
COMPANY	CO	Enter company name/address.	10.4
DATE	D	Enter date.	10.5
FINISH	F	Finish transmittal file creation.	10.6
HELP	H	List command summary.	10.7
LIST	L	List transmittal file.	10.8
NAME	N	Enter responsible person name/phone.	10.9
OPTION	{ O[<opt#>] <opt#> }	Enter chip options.	10.10
PRINT	P <chip#>	Print available options for chip.	10.11
TRANSMITTAL	T <filename>	Begin creation of transmittal file.	10.12

Where: <opt#> = Number of valid options for current chip number. "0" may be left off of command call of <opt#> is used. This number causes entry mode to be entered at the specified option number. If "0" alone is used, entry is at the beginning of the option list.

<chip#> = Valid chip number and letter.

<filename> = Standard PDS filename.

ROM VALUES

000	00	33	5E	33	6C	2E	8D	3E	8D	91	3A	70	3E	7D	33	A8
010	7F	33	B8	7F	2E	7D	61	80	00	01	51	11	51	03	51	13
020	51	5E	49	48	00	00	00	00	00	00	00	00	00	00	00	00
030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
040	33	B8	15	5F	CC	5F	DA	5B	68	60	63	C6	00	5F	21	F1
050	2C	05	5F	00	26	50	00	16	72	CA	00	58	21	EF	91	CA
060	2C	05	52	5F	48	06	25	50	23	28	16	23	38	06	48	00
070	52	55	21	CA	3A	46	CA	00	00	00	00	00	00	00	00	00
080	15	23	B9	05	23	A9	48	05	23	B9	05	04	83	00	07	8D
090	48	0E	68	8D	1D	00	52	07	95	1E	70	70	2C	70	48	3C
0A0	33	2A	04	06	4C	32	4F	5F	4D	C8	05	51	51	5F	AB	48
0B0	3F	04	04	04	04	04	04	04	07	0E	33	3E	48	22	00	56
0C0	30	4A	07	00	56	30	4A	06	05	48	00	00	00	00	00	00
0D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
100	43	01	4B	03	4B	03	01	03	00	4B	4B	30	02	14	24	03
110	01	23	21	03	49	02	90	A0	B4	54	93	02	24	01	A0	02
120	00	CA	08	4B	4B	D8	3B	10	30	B4	FC	48	80	00	C2	90
130	4A	48	0A	4A	48	42	42	48	4A	4A	CE	88	10	02	04	41
140	5F	F7	8F	39	0F	79	71	BD	F6	09	11	70	38	36	36	3F
150	F3	3F	F3	ED	01	3E	30	36	00	00	09	31	00	0E	00	08
160	00	00	20	FF	ED	ED	58	00	00	00	C0	C0	00	C0	00	00
170	31	00	51	41	60	61	71	01	71	61	01	00	80	C8	04	83
180	1E	15	54	BF	33	2C	16	06	0F	BF	33	2C	16	06	38	15
190	33	3C	33	5F	1F	22	05	B9	4F	44	0F	05	4F	44	1E	05
1A0	4F	0E	05	3E	4F	35	50	32	4F	41	ED	6A	80	BA	33	5F
1B0	3E	35	AB	50	05	23	8F	15	23	80	05	1E	06	43	42	9F
1C0	6B	40	33	5E	3E	05	52	D0	23	3D	2A	17	05	5C	DB	D7
1D0	51	DE	23	3D	2B	68	B8	A9	A9	32	F4	6B	4D	FB	23	3D
1E0	5F	E8	2E	05	5E	E9	63	C0	A9	2D	05	3E	21	D8	3D	05
1F0	3C	32	21	22	68	18	32	2E	00	30	06	3E	AA	06	61	80

ROM VALUES

200	30	31	32	33	34	35	36	37	38	39	30	2A	2D	00	00	FF
210	00	7D	51	57	45	52	54	59	55	40	4F	50	0A	00	00	00
220	00	41	53	44	46	47	48	4A	4B	4C	3B	7F	0D	00	00	00
230	00	5A	58	43	56	42	4E	4D	2C	2E	2F	20	08	00	00	00
240	00	21	22	23	24	25	26	27	28	29	40	3A	3D	00	00	00
250	00	7D	51	57	45	52	54	59	55	49	5F	40	0A	00	00	00
260	00	41	53	44	46	47	48	4A	4B	4C	3B	7F	0D	00	00	00
270	00	5A	58	43	56	42	4E	4D	2C	2E	2F	20	08	00	00	00
280	33	A1	05	5F	C7	06	F0	07	C2	3A	11	CD	D6	33	A2	25
290	16	73	35	4E	58	CF	2F	7D	7A	33	A7	BD	5A	F4	07	BD
2A0	5A	F0	07	BD	5E	ED	33	A3	05	5C	ED	07	70	2F	7C	77
2B0	3E	05	50	48	33	A7	01	C0	F0	00	00	00	00	00	00	00
2C0	0D	00	07	C2	0F	06	1D	00	52	07	C9	1F	06	48	22	00
2D0	2B	11	32	03	D6	13	54	3D	13	53	03	52	11	51	2D	BF
2E0	33	A8	33	2C	16	06	20	42	48	00	00	00	00	00	00	00
2F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
300	0A	0D	0F	13	18	2B	38	3A	35	35	33	B8	D6	2A	DF	29
310	43	4C	DD	29	35	50	80	F5	3B	05	5E	E6	06	05	50	87
320	F5	33	B8	05	5E	D6	28	7F	38	7F	F5	2C	05	5F	E1	06
330	33	91	80	6A	C0	33	6C	48	91	E6	29	15	70	06	63	0A
340	33	01	48	33	68	39	13	DF	29	33	2C	16	06	39	05	56
350	DD	15	23	B8	05	23	A8	68	60	39	76	63	26	00	FF	01
360	E6	73	29	25	50	C9	72	29	43	4D	05	50	33	2C	07	CC
370	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
380	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
390	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
3A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
3B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
3C0	9F	5F	C6	51	68	18	61	FB	B9	3E	05	2D	06	3C	05	3D
3D0	06	2E	70	3A	03	C6	6A	CE	3B	13	E9	05	52	06	23	.28
3E0	B0	23	38	B0	3A	01	60	40	C6	3F	4B	E4	00	00	00	00
3F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

SOURCE CHECKSUM 62A0  
 OBJECT CHECKSUM 102B  
 TRANSMIT CHECKSUM B8A7  
 M> F  
 FINISH CREATION OF TRANSMITTAL FILE  
 (Y/N, CR=YES)? CR  
 DISK TO BE MAILED IN DRIVE  
 (Y/N, CR=YES)? CR  
 CREATING FILE JOEUSER:MASKEX.TRN  
 M>

The disk is now ready to be sent to:  
 National Semiconductor Corp.  
 2900 Semiconductor Drive  
 Santa Clara, CA 95051  
 ATTN:  
 COP Control Customer Service  
 DISK/DISK/DISK/DISK/DISK/DISK

Note: A mailing package, which includes a label with this information, is available from:  
 COPS Marketing, MS C2385  
 National Semiconductor Corp.  
 2900 Semiconductor Drive  
 Santa Clara, CA 95051  
 Phone: (408) 737-5883

# Memory Diagnostic

## 11.1 Use of Memory Diagnostic (MDIAG)

MDIAG allows the user to run diagnostics on the PDS memory. This program will run ADDRESS, BIT, WORD, and GALPAT tests. The program will test the area in which it resides by moving the entire program just before the tests are run on that section of memory. After the tests are run on that section of memory, the program is restored to its original location.

All reports of errors or passes are sent to the console unless the \*PR is invoked, in which case all reports will be routed to the printer (i.e., overnight runs).

### 11.1.1 ADDRESS Test

In the ADDRESS test, the address of each memory location in the test range is stored in that memory location. Each location is then checked for the proper value. This test checks the addressing capability of the PDS CPU and MEMORY boards.

### 11.1.2 WORD Test

In the WORD test, a value is stored in successive memory locations. As each value is stored, that memory location is checked for the proper value. The value is then complemented, stored again, and the location is rechecked. Finally, the value is re-complemented, stored again, and the location is rechecked once again. Then the next memory location is tested in the same manner, until the end of the test range is reached. Then the entire test is repeated for a total of two passes. This test checks whether memory words within the test range can save the given values and their complements.

### 11.1.3 BIT Test

In the BIT test, each bit of each word in the test range is tested in the same manner as in the WORD test above. This test is identical to the WORD test except that a single "1"-BIT MASK is used for memory store and compare operations, and this mask is changed (after completing testing of the bit) to the next bit of the word. When all bits of one word are exhausted, the test advances to the next word of the test range.

### 11.1.4 GALPAT Test

In the GALPAT test, a background word (X'AAAA) is stored in each memory location in the test range. Each location in the test range is then checked for the proper value. Next, a test word (X'5555) is stored in the first memory location of the test range. All the background locations are tested sequentially, with the test location being tested between each sequential background location.

Then, the location where the test word was loaded is restored to the background word, and the test word is moved to the next location following the one it was stored in previously. All locations in the test range are checked again, in the same manner. This process continues until the test word has been stored once in every location of the test range. This completes PASS 1.

Then, the background word and the test word are swapped (background = X'5555, test word = X'AAAA) and the entire test is repeated for PASS 2. This test checks for "crosstalk" between memory locations. Note that the test time is proportional to the square of the range to be tested. Testing a 2k range takes about four times as long as testing a 1k range.

The diagnostics are broken into ADDRESS/WORD/BIT tests on 0/4k, 4/16k, A000/AFFF, and DC00/DFFF. The GALPAT tests are broken into 1k increments. The parameter routine prompts for all required inputs: addresses to be tested, tests to be run, and the mode in which the tests are to be run. The tests allowed are ADDRESS (A), WORD (W), BIT (B), GALPAT (G), or ALL (CR).

The modes allowed are CONTINUOUS (C) and HALT (H).

The CONTINUOUS (C) MODE continues testing until interrupted by a keyboard entry. If an error is encountered, the error is reported and the next block of memory is then tested. If no errors occur, then the block tested is reported and the next block is tested.

The HALT (H) MODE tests the memory until there is an error, in which case the error is reported and the test is terminated, or until the entire requested test range is tested, in which case the addresses are reported and the test halts.

To call MDIAG, type:

```
X> @MDIAG
    MDIAG, REV A., DATE
    M>
```

MDIAG is then ready to accept one of the commands described below.

The commands accepted by this program are:

**PARAMETER** — This command gets all the parameters required to run this program. The user is prompted for the type of input required and illegal responses are rejected.

**RUN**: This command runs the test as specified by the input to the PARAMETER command.

Examples:

```
M> PA
ADDRESS RANGE 0/3FFF, A000/AFFF, DC00/DFFF
MAY BE SPECIFIED 0/AFFF OR 100/FFFF ETC.
ADDRESS RANGE TO BE TESTED? (CR = ALL)
0/3FFF
TESTS? (CR = YES) A, B, W
MODE FOR RUNNING TESTS, C = CONTINUOUS,
H = HALT
MODE (CR = H) C
M> RUN
ADDRESS, WORD, BIT TEST(S) PASSED AT 1000/3FFF
ADDRESS, WORD, BIT TEST(S) PASSED AT 0000/0FFF
ADDRESS, WORD, BIT TEST(S) PASSED AT 1000/3FFF
ADDRESS, WORD, BIT TEST(S) PASSED AT 0000/0FFF
```



CONTINUE TEST (Y/N, CR = YES)? CR  
(keyboard interrupt)

ADDRESS, WORD, BIT TEST(S) PASSED AT 1000/3FFF  
CONTINUE TEST (Y/N, CR = YES)? N  
M>

This example ran the ADDRESS, WORD, and BIT tests on the system program memory space.

M> PA  
ADDRESS RANGES 0/3FFF, A000/AFFF, DC00/DFFF  
MAY BE SPECIFIED 0/AFFF OR 100/FFFF ETC.  
ADDRESS RANGE TO BE TESTED? (CR = ALL)  
A000/A080  
TEST TO BE RUN  
A = ADDRESS, B = BIT, W = WORD, G = GALPAT  
TESTS? (CR = ALL) CR

MODE FOR RUNNING TESTS, C = CONTINUOUS,  
H = HALT

MODE? (CR = H) C  
M> RU

ADDRESS, WORD, BIT TEST(S) PASSED AT A000/A080  
GALPAT BACKGROUND ERROR TEST FAILED AT  
A010  
DATA SHOULD BE AAAA/DATA IS AAEA  
ADDRESS, WORD, BIT TEST(S) PASSED AT A000/A080  
CONTINUE TEST (Y/N, CR = YES)? N  
M>

This example runs all tests on addresses A000/A080 reporting pass/fail information until a keyboard interrupt. (NOTE: A KEYBOARD INTERRUPT IS ONLY TESTED DURING A MESSAGE OUTPUT.)

# COP400 PDS PROM Programmer (PROG)

## 12.1 Introduction

PROG operates the PROM programmer located in the center of the PDS front panel. PROG programs MM2716, MM2732, MM2724 and MM2758 EPROMs.

The PROG program, using a COP Load Module, programs the EPROMs. The PROM is used on an emulator board with a COP400 ROMless device. PROG uses PDS Shared Memory as the data buffer. When the COP400 program has been developed using COPMON, and is in shared memory, it can be saved on a PROM by using PROG.

Shared Memory is divided into blocks, corresponding to the size of the PROM. For example, if 1k PROMs are used, there are four blocks of 1k each.

- Block 0 is 0 — X'3FF of shared memory.
- Block 1 is 400 — 7FF of shared memory.
- Block 2 is 800 — BFF of shared memory.
- Block 3 is C00 — FFF of shared memory.

The block operand in the Program, Dump and Compare commands specifies which part of shared memory to use.

Tables 12-1 and 12-2 contain a command and operand summary, respectively.

## 12.2 ALTER Data Buffer Command

Syntax: ALTER [<addr>],[<value>]. . .

Changes the contents of consecutive data buffer locations to the specified hexadecimal values beginning at the specified address. Consecutive commas will increment the current address pointer, leaving the data at these locations unaltered.

Example:

```
P>A 10,60,,44
```

Place 60 in location 10 and leave 11 unchanged, and place 44 in location 12.

## 12.3 BASE Command

Syntax: BASE

Displays the base address used in the last LOAD command.

Example:

```
P>BA
```

```
CURRENT LOAD BASE = 1000
```

## 12.4 CHIP Command

Syntax: CHIP [<chip#>]

Displays and changes the PROM that the system is configured for dumping and programming. The user enters the number of the PROM to be programmed (see Table 12-1). If no number is specified, the current number is displayed.

Example:

```
P>CH 32
```

Sets up programmer for MM2732's.

## 12.5 COMPARE Command

Syntax: COMPARE [<block#>]

Compares the contents of the data buffer block with the contents of the PROM. Default block is 0.

Example:

```
P>CO
```

```
COMPARE DONE
```

## 12.6 DEPOSIT Command

Syntax: DEPOSIT<value>[,<addr range>]

Copies the specified value to each location specified in the address range in the buffer. Default is every location in the buffer.

Table 12-1. PROM Programmer Command Summary

Command	Syntax	Description	Section
ALTER	A [<addr>],[<value>]. . .	Alter data in buffer location specified.	12.2
BASE	BA	Display base address.	12.3
CHIP	CH [<chip#>]	Display/change PROM chip number.	12.4
COMPARE	CO [<block#>]	Compare buffer to PROM.	12.5
DEPOSIT	DE <value>[,<addr range>]	Copies specified value to buffer.	12.6
DUMP	DU [<block#>]	Dumps PROM contents into buffer.	12.7
ERASE	E [Y/N]	Verifies whether the PROM is erased or not.	12.8
HELP	H	Displays command summary.	12.9
LIST	L [<addr>]	Lists contents of specified location.	12.10
LOAD	<filename>[.LM][<base addr>]	Loads file from diskette into buffer.	12.11
PROGRAM	[<block#>][,<addr>]	Programs the PROM with specified buffer.	12.12

Example:

```
P>DE FF, 0/FF
```

Copies X'0FF in to the buffer location 0 to FF.

### 12.7 DUMP Command

Syntax: DUMP [<block#>]

Dumps the PROM into the buffer block specified. Default is block 0.

Example:

```
P>DU
```

### 12.8 ERASE Command

Syntax: ERASE [Y|N]

Verifies that the EPROM is erased before programming. Default is report status of erase flag.

Example:

```
P>E N
```

```
DO NOT CHECK FOR ERASED BEFORE
PROGRAMMING
```

### 12.9 HELP Command

Syntax: HELP

Prints out the command summary.

Example:

```
P>H
```

```
ALTER (A) [<ADDRESS>],<VALUE>[,<VALUE>]
BASE (BA)
CHIP (CH)<CHIP#>
  WHERE CHIP# :: = 16/58A/58B/32/24A/24B
COMPARE (C) [<BLOCK#>]
DEPOSIT (DE)<VALUE>[,<RANGE>]
DUMP (D) [<BLOCK#>]
ERASE (E) [<Y/N>]
HELP (H)
LIST (L) [<RANGE>]
LOAD (LO)<FILENAME>[BASE ADDRESS]
PROGRAM (P) [<BLOCK#>][,PROGRAM RANGE]
```

### 12.10 LIST Command

Syntax: LIST [<addr>]

Hex lists (hexadecimal) the contents of each location in the specified address range. Default is current address.

Example:

```
P>L 0/5
```

```
000 00 44 60 33 51 0F
```

### 12.11 LOAD Command

Syntax: LOAD <filename>[.LM] [<base address>]

Loads file from disk into buffer area. The base address option has been implemented to enable users to deal with programs larger than 4k bytes. A 4k byte segment of the program, starting at <base address>, is loaded into shared memory.

For example, if the program MYPROG occupies absolute addresses X'3000 to X'4BFF then the command LO MYPROG, 3100 will load the segment of the program from X'3100 to X'0FF into locations 0 through X'FF of shared memory.

Example:

```
P>LO MYFILE
```

```
FINISHED LOADING
```

### 12.12 PROGRAM Command

Syntax: PROGRAM<block#>][,<addr>]

Programs the PROM from the buffer clock specified, default is block 0. The range option allows the user to program single bytes or a range of bytes within the PROM.

Example:

```
P>P
```

```
INSERT 2716, PROGRAM (Y/N, CR = YES)? CR
PROGRAMMING
VERIFYING
CKSM = 0123
```

Sample Program Session:

```
X>@PROG
```

```
PROG, REV:A,MAY 21 1981
```

```
ALTER (A) [<ADDRESS>],<VALUE>[,<VALUE>]
```

```
BASE (BA)
```

```
CHIP (CH) <CHIP#>
```

```
  WHERE CHIP# :: = 16/58A/58B/32/24A/24B
```

```
COMPARE (C) [<BLOCK#>]
```

```
DEPOSIT (DE)<VALUE>[,<RANGE>]
```

```
DUMP (D) [<BLOCK#>]
```

```
ERASE (E) [<Y/N>]
```

```
HELP (H)
```

```
LIST (L) [<RANGE>]
```

```
LOAD (LO)<FILENAME>[BASE ADDRESS]
```

```
PROGRAM (P) [<BLOCK#>][,PROGRAM RANGE]
```

```
P>LO MYFILE
```

```
FINISHED LOADING
```

```
P>CH 16
```

```
P>P
```

```
INSERT 2716, PROGRAM (Y/N, CR = YES)? CR
```

```
PROGRAMMING
```

```
VERIFYING
```

```
CKSM = 4FBD
```

```
P>
```

Table 12-2. Summary of PROG Operands

Operand	Description
<addr>	One to three hexadecimal digits. 0-0FF. P— Address prior to current address . — Current address N— Next address after current address L— Last address in buffer
<addr range>	<addr>[<addr>]
<base address>	One to four hexadecimal digits. 0-0FFFF.
<block#>	This depends on the chip specified. MM2716: 0-1 MM2758: 0-3 MM2732: 0 MM2724: 0-1
<chip#>	This depends on the chip specified. MM2716: 16 MM2758A: 58A MM2758B: 58B MM2732: 32 MM2724A: 24A MM2724B: 24B
<filename>	Valid name of COP400 LM file.
N	NO— Do not check for erase before programming.
Y	YES— Check for erase before programming.
<value>	Hexadecimal number in the range 0-FF.

## Appendix A Sample Program

This appendix describes the creation, assembly, and debugging of a COP program on the COP400 Product Development System.

The user can enter a COP program using EDIT. The program to be created here will read a number from the COP420 I lines and add 5. The carry will be ignored. The result will be output on the D outputs, and the decoded 7-segment equivalent will appear on the L outputs. A 50% duty cycle square wave will appear on the SK output. The pulse width will increase with the magnitude of the above addition. As the user changes the data on the I inputs, there should be corresponding changes on the other outputs. These outputs may be examined and verified on an oscilloscope. The probes may be attached directly to the proper pins on the COP output cable from the emulator card. The program is called COPEX.

```
F>@EDIT COPEX
EDIT,REV:B
CREATE NEW FILE (Y/N, CR = YES)? CR
AVAILABLE SECTORS: 496
E>I
  1?      .TITLE COPEX, 'COP EXAMPLE'
  2?      CLRA
  3?      LEI 5          ;Q TO L, C TO SK ON XAS
  4? START:
  5?      ININ          ;READ 10-13 TO A
  6?      AISC 5        ;ADD 5
  7?      OBD           ;OUTPUT A TO D0-D3
  8?      LB#          (ABORTED LINE TO INSERT A NOP AFTER THE AISC)
  8? CR
E>I TO 7
  7?      NOP
  8? CR
E>I
  9?      LBI 0         ;SAVE ENTERED VALUE +5 IN
 10?      X             ;MO
 11?      CLRA         ;SET UP FOR
 12?      AISC 4        ;LQID ON PAGE 1
 13?      LQID         ;PERFORM SEGMENT LOOKUP
 14?      RC
 15?      XAS          ;OUTPUT 0 TO SK
 16?      NOP
 17?      NOP          ;DELAY FOR 50% DUTY CYCLE
 18?      NOP
 19?      NOP
 20?      NOP
 21?      NOP
 22?      NOP
 23?      NOP
 24?      NOP
 25?      NOP
 26?      NOP
 27?      NOP
 28?      NOP
 29?      NOP
 30?      COMP         ;MAKE DELAY PROPORTIONAL
 31?      AISC 1        ;TO VALUE +5
 32?      JP .-1
 33?      SC
 34?      XAS          ;OUTPUT 1 TO SK
 35?      LD           ;GET ENTERED VALUE +5
 36?      COMP         ;DELAY PROPORTIONAL TO
 37?      AISC 1        ;ENTERED VALUE +5
 38?      .JP .-1
 39?      JP START
```

```

40?      .PAGE 1
41?      .WORD 03F,006,05B,04F,066,06D,07D
42?      .WORD 007,07F,067;0-9
43?      .WORD 077,07C,039,05E,079,071;A-F
44?      .END
45? CR   (EXIT INPUT MODE)

```

```

E>FI
FINISH CURRENT EDIT (Y/N, CR = YES)? CR

```

The user may verify the new program on the disk by displaying the directory with FM.

```

E>@FM
FM,REV:B
F>D
DIRECTORY FOR: PDSUSER "PDS USER"

```

FN	D NAME	TYPE	SIZE	PL	VN
1	EDIT	.MP MAIN PROGRAM	20	2	3
2	ASM	.MP MAIN PROGRAM	32	2	3
3	COPMON	.MP MAIN PROGRAM	32	2	3
4	FM	MP MAIN PROGRAM	16	2	3
5	DIKIT	.MP MAIN PROGRAM	12	2	3
6	COPEX	.SRC SYMBOLIC	4	2	3

```

SECTORS BAD:      0
SECTORS USED:    124
SECTORS FREE:    492

```

The user may not assemble the COP program, displaying the assembly errors on the console.

```

F>@ASM
ASM,REV:C
A>I= COPEX,0 = COPEX,L = *CN,EL
CREATING FILE PDSUSER:COPEX.LM
END PASS 1
COP CROSS ASSEMBLER PAGE 1
COPEX COP EXAMPLE
13 00D 00      LQUID          ;PERFORM SEGMENT LOOKUP
ERROR UNDEFINED @
1 ERROR LINES
56 ROM WORDS USED
END PASS 4
SOURCE CHECKSUM = E88F
OBJECT CHECKSUM = 0276
INPUT FILE      PDSUSER:COPEX.SRC
OBJECT FILE     PDSUSER:COPEX.LM
A>

```

The above assembly error ("LQUID" should be "LQID") can be edited with EDIT.

```

A>@EDIT COPEX
EDIT,REV:B
AVAILABLE SECTORS: 488
INPUT FILE SECTORS: 4
E>RE
EOF AT 44
E>10/L

```

```

10          X
11          CLRA          ;SET UP A FOR LQID ON
12          AISC          4          ;PAGE 1
13          LQID          ;PERFORM SEGMENT LOOKUP
14          RC
15          XAS          ;OUTPUT 0 TO SK
16          NOP
17          ***
E>

```

The listing was interrupted by the user pressing a key when the error was located. The "LQID" is replaced by a "LQID."

```

E>E 13
13          LQID          ;PERFORM SEGMENT LOOKUP
EDITS?     LQID          ;PERFORM SEGMENT LOOKUP
13          LQID          ;PERFORM SEGMENT LOOKUP
EDITS? CR
E>FI
FINISH CURRENT EDIT (Y/N, CR = YES)? CR
OK TO DELETE FILE PDSUSER:COPEX.SRC (Y/N, CR = YES)? CR
E>

```

The user may now re-assemble the corrected program, obtaining an assembly load module file (COPEX.LM) and a full assembly output listing.

```

E>@ASM I = COPEX,0 = COPEX,L = *PR
ASM,REV:C
OK TO DELETE FILE PDSUSER:COPEX.LM (Y/N, CR = YES)? CR
CREATING FILE PDSUSER:COPEX.LM
END PASS 1
END PASS 4
A>

```

Notice that the listing was assigned to the printer. The printer listing is shown below. No assembly errors occurred.

```

COP CROSS ASSEMBLER PAGE 1
COPEX COP EXAMPLE
1          .TITLE COPEX, 'COP EXAMPLE'
2 000     00    CLRA
3 001     3365  LEI      5          ;Q TO L, C TO SK ON XAS
4          START:
5 003     3328  ININ          ;READ 10-13 TO A
6 005     55    AISC      5          ;ADD 5
7 006     44    NOP
8 007     333E  OBD          ;OUTPUT A TO D0-D3
9 009     0F    LBI      0          ;SAVE ENTERED VALUE +5
10 00A     06    X
11 00B     00    CLRA          ;SET UP A FOR LQID ON
12 00C     54    AISC      4          ;PAGE 1
13 00D     BF    LQID          ;PERFORM SEGMENT LOOKUP
14 00E     32    RC
15 00F     4F    XAS          ;OUTPUT 0 TO SK
16 010     44    NOP
17 011     44    NOP          ;DELAY FOR 50% DUTY CYCLE
18 012     44    NOP
19 013     44    NOP
20 014     44    NOP
21 015     44    NOP
22 016     44    NOP
23 017     44    NOP
24 018     44    NOP
25 019     44    NOP

```

```

26 01A 44 NOP
27 01B 44 NOP
28 01C 44 NOP
29 01D 44 NOP
30 01E 40 COMP ;MAKE DELAY PROPORTIONAL
31 01F 51 AISC 1 ;TO ENTERED VALUE +5
32 020 DF JP -1
33 021 22 SC
34 022 4F XAS ;OUTPUT 1 TO SK
35 023 05 LD ;DELAY PROPORTIONAL TO
36 024 40 COMP ;ENTERED VALUE +5
37 025 51 AISC 1
38 026 E5 JP -1
39 027 C3 JP START
40 0040 .PAGE 1
41 040 3F .WORD 03F,006,05B,04F,066,06D,07D
041 06
042 5B
043 4F
044 66
045 6D
046 7D
42 047 07 .WORD 007,07F,067;0-9
048 7F
049 67

```

COP CROSS ASSEMBLER PAGE 2

COPEX COP EXAMPLE

```

43 04A 77 .WORD 077,07C,039,05E,079,071;A-F
04B 7C
04C 39
04D 5E
04E 79
04F 71
44 .END

```

COP CROSS ASSEMBLER PAGE 3

COPEX COP EXAMPLE

START 0003

NO ERROR LINES

56 ROM WORDS USED

SOURCE CHECKSUM = E85A

OBJECT CHECKSUM = 027C

INPUT FILE PDSUSER:COPEX.SRC

OBJECT FILE PDSUSER:COPEX.LM

The new program may be tested now using COPMON. The chip number is 420. To make it easier to see the program, shared memory is zero-filled before loading the new program COPEX.

A> @COPMON

COPMON,REV:C

CHIP NUMBER (DEFAULT = 420)? CR

C>DE 0,0/L

C>LO COPEX

FINISHED LOADING

To begin execution of the program, first reset the COP, then start by giving the 'GO' command.

C>RE

CHIP IS RESET

C>G



On examining the outputs, it is discovered that the L outputs have the proper values, but the D lines do not. Also, the square wave on the SK line is incorrect. Only one half of the cycles varies with the input. Begin by obtaining a trace and examine the path of the COP device.

```

C>TR
TRACE ENABLED:
  A:001 OCCUR: 1 PRIOR: 0 GO:N
C>RE
CHIP IS RESET
C>G
TRACED ON A:001 AT A:001
C>I
  0  0 A:001      E:1111
  1  1 A:002      SKIP E:1111
  2  2 A:003      E:1111
  3  3 A:004      SKIP E:1111
  4  4 A:005      E:1111
  5  5 A:006      SKIP E:1111
  6  6 A:007      E:1111
  7  7 A:008      SKIP E:1111
  8  8 A:009      E:1111
  9  9 A:00A      E:1111
 10 10 A:00B      E:1111
 11 11 A:00C      E:1111
 12 12 A:00D      E:1111
 13 13 A:044      SKIP E:1111
 14 14 A:00E      E:1111
 15 15 A:00F      E:1111

C>I
 16 16 A:010      E:1111
 17 17 A:011      E:1111
 18 18 A:012      E:1111
 19 19 A:013      E:1111
 20 20 A:014      E:1111
 21 21 A:015      E:1111
 22 22 A:016      E:1111
 23 23 A:017      E:1111
 24 24 A:018      E:1111
 25 25 A:019      E:1111
 26 26 A:01A      E:1111
 27 27 A:01B      E:1111
 28 28 A:01C      E:1111
 29 29 A:01D      E:1111
 30 30 A:01E      E:1111
 31 31 A:01F      E:1111

C>I
 32 32 A:020      E:1111
 33 33 A:01F      E:1111
 34 34 A:020      E:1111
 35 35 A:01F      E:1111
 36 36 A:020      E:1111
 37 37 A:01F      E:1111
 38 38 A:020      E:1111
 39 39 A:01F      E:1111
 40 40 A:020      E:1111
 41 41 A:01F      E:1111
 42 42 A:020      E:1111
 43 43 A:01F      E:1111
 44 44 A:020      E:1111
 45 45 A:01F      E:1111
 46 46 A:020      E:1111
 47 47 A:01F      E:1111

```

```

C>T
48 48 A:020      E:1111
49 49 A:01F      E:1111
50 50 A:020      E:1111
51 51 A:01F      E:1111
52 52 A:020      E:1111
53 53 A:01F      E:1111
54 54 A:020      E:1111
55 55 A:01F      E:1111
56 56 A:020      E:1111
57 57 A:01F      E:1111
58 58 A:020      E:1111
59 59 A:01F      E:1111
60 60 A:020      E:1111
61 61 A:01F      E:1111
62 62 A:020      SKIP E:1111
63 63 A:021      E:1111

```

The word SKIP indicates that the instruction was skipped. It also appears on the second half of 2-word instructions. Notice that at trace location 13, the address is 44. This is actually the second half of the LQID instruction, and is the address of the data to be loaded into the Q register. The second instruction, LEI 5, assigns the Q register to the L outputs. According to the trace, program execution has proceeded as expected, except that the loop at locations 1F and 20 was done 15 times. Examination of the listing at those locations shows that the accumulator wasn't loaded with the entered value before the first loop. The LD instruction before the COMP instruction was omitted. Single-stepping through the first several locations allows the user to inspect the COP registers, particularly the accumulator and the B register.

```

C>R
CHIP IS RESET
C>AU ALL
C>S
STEP A:0 B:00 C:0 G:0 I:F L:FF Q:66 S:F P:001
M0:FFFFFFFFFFFFFFFFFA M1:FFFFFFFFFFFFFFFFFF
M2:9FFFFFFFFFFFFFFFFF M3:3377777777777777
C>CR
STEP A:0 B:00 C:0 G:0 I:F L:66 Q:66 S:F P:003
M0:FFFFFFFFFFFFFFFFFA M1:FFFFFFFFFFFFFFFFFF
M2:9FFFFFFFFFFFFFFFFF M3:3377777777777777
C>CR
STEP A:F B:00 C:0 G:0 I:F L:66 Q:66 S:F P:005
M0:FFFFFFFFFFFFFFFFFA M1:FFFFFFFFFFFFFFFFFF
M2:9FFFFFFFFFFFFFFFFF M3:3377777777777777
C>CR
A:006 SKIPPED
STEP A:4 B:00 C:0 G:0 I:F L:66 Q:66 S:F P:007
M0:FFFFFFFFFFFFFFFFFA M1:FFFFFFFFFFFFFFFFFF
M2:9FFFFFFFFFFFFFFFFF M3:3377777777777777
C>CR
STEP A:4 B:00 C:0 G:0 I:F L:66 Q:66 S:F P:009
M0:FFFFFFFFFFFFFFFFFA M1:FFFFFFFFFFFFFFFFFF
M2:9FFFFFFFFFFFFFFFFF
C>

```

From looking at the assembly listing, one sees that location 7 has the OBD instruction which puts the B register contents out to the D lines. After executing this instruction, B still contains zero but A contains the correct value. A CAB instruction is necessary before the OBD. Both of the mistakes in this program require instructions to be inserted when it is edited. But the NOP at location 10 may be easily replaced with an LD instruction, giving a much better square wave. After starting the chip, the square may be displayed again.

```

C> PU 1D,LD
C> CL
BRKPT AND TRACE CLEARED
C> G
C>

```

The program may now be re-edited.

```

C> @EDIT COPEX
EDIT,REV:B
AVAILABLE SECTORS: 480
INPUT FILE SECTORS: 4
E> RE
EOF AT 44
E> L
 1          .TITLE COPEX, 'COP EXAMPLE'
 2          CLRA
 3          LEI      5          ;Q TO L, C TO SK ON XAS
 4  START:
 5          ININ     ;READ 10-13 TO A
 6          AISC    5          ;ADD 5
 7          NOP
 8          OBD     ;OUTPUT A TO D0-D3
 9          LBI     0          ;SAVE ENTERED VALUE +5
10          X
11         CLRA     ;SET UP A FOR
12#****
E>

```

The missing CAB instruction should be inserted to line 8.

```

E> IN TO 8
 8?          CAB
 9? CR
E> L
 1          .TITLE COPEX, 'COP EXAMPLE'
 2          CLRA
 3          LEI      5          ;Q TO L, C TO SK ON XAS
 4  START:
 5          ININ     ;READ 10-13 TO A
 6          AISC    5          ;ADD 5
 7          NOP
 8          CAB
 9          OBD     ;OUTPUT A TO D0-D3
10         LBI     0          ;SAVE ENTERED VALUE +5
11         X
12         CLRA     ;SET UP A FOR
13         AISC    4          ;LQID#****
E> L 25
 25         NOP
E> N 21
 26         NOP
 27         NOP
 28         NOP
 29         NOP
 30         NOP
 31         COMP     ;MAKE DELAY PROPORTIONAL
 32         AISC    1          ;TO ENTERED VALUE +5
 33         JP      -1

```

```

34      SC
35      XAS          ;OUTPUT 1 to SK
36      LD          ;G#***
E>

```

The missing LD instruction should be inserted to line 31.

```

E>IN TO 31
  31?   LD          ;GET ENTERED VALUE +5
  32? CR
E>L 25
  25     NOP
E>N 21
  26     NOP
  27     NOP
  28     NOP
  29     NOP
  30     NOP
  31     LD          ;GET ENTERED VALUE +5
  32     COMP       ;MAKE DELAY PROPORTIONAL
  33     AISC      1  ;TO ENTERED VALUE +5
  34     JP        . -1
  35     SC
  36     XAS          #***
E>

```

The edit mode may be finished now, replacing the old program with the new one.

```

E>FI
FINISH CURRENT EDIT (Y/N, CR = YES)? CR
OK TO DELETE FILE PDSUSER:COPEX.SRC (Y/N, CR = YES)? CR
E>

```

The new program may be verified by re-assembling and testing with COPMON.

```

E>@ASM I = COPEX.0 = COPEX.L = *PR
ASM,REV:C
OK TO DELETE FILE PDSUSER:COPEX.LM (Y/N, CR = YES)? CR
CREATING FILE PDSUSER:COPEX.LM
END PASS 1
END PASS 4
A>

```

The new assembled program may be tested with COPMON and an oscilloscope as before to verify proper performance.

```

A>@COPMON
COPMON,REV:C
CHIP NUMBER (DEFAULT = 420)? CR
C>LO COPEX
FINISHED LOADING
C>RE
CHIP IS RESET
C>G
C>

```

Now that both the source and load module files are correct, the deleted versions may be packed with the commands in file manager, giving more room on the disk for new programs.

```
C>@FM
FM,REV:B
F>
```

Now the new disk is examined and packed.

```
F>D
```

```
DIRECTORY FOR: PDSUSER "PDS USER"
```

FN	D NAME	TYPE	SIZE	PL	VN
1	EDIT	.MP MAIN PROGRAM	20	2	3
2	ASM	.MP MAIN PROGRAM	32	2	3
3	COPMON	.MP MAIN PROGRAM	32	2	3
4	FM	.MP MAIN PROGRAM	16	2	3
5	DSKIT	.MP MAIN PROGRAM	12	2	3
	* COPEX	.SRC SYMBOLIC	4	2	1
	* COPEX	.LM LOAD MODULE	4	2	1
	* COPEX	.SRC SYMBOLIC	4	2	2
	* COPEX	.LM LOAD MODULE	4	2	2
6	COPEX	.SRC SYMBOLIC	4	2	3
7	COPEX	.LM LOAD MODULE	4	2	3

```
SECTORS BAD:      0
SECTORS USED:    144
SECTORS FREE:    472
```

```
F>P
```

```
PACKING DISK (Y/N,CR = YES)? CR
```

```
F>D
```

```
DIRECTORY FOR: PDSUSER "PDS USER"
```

FN	D NAME	TYPE	SIZE	PL	VN
1	EDIT	.MP MAIN PROGRAM	20	2	3
2	ASM	.MP MAIN PROGRAM	32	2	3
3	COPMON	.MP MAIN PROGRAM	32	2	3
4	FM	.MP MAIN PROGRAM	16	2	3
5	DSKIT	.MP MAIN PROGRAM	12	2	3
6	COPEX	.SRC SYMBOLIC	4	2	3
7	COPEX	.LM LOAD MODULE	4	2	3

```
SECTORS BAD:      0
SECTORS USED:    128
SECTORS FREE:    488
```

# COP400 In-System Emulator™ Boards

## User's Manual



Publication No. 420306469-001A  
Order No. 420306469-001  
June 1981

# Preface

This manual describes the COP400 In-System Emulator (ISE™) Boards. The In-System Emulator Boards allow emulation of COP400 devices.

These boards are designed to stand-alone or to be used in conjunction with a Program Development System (PDS) or STARPLEX™ Development System.

The manual (Publication No. 420306469) supercedes Publication No. 420306143-001 in all revisions.

The material presented in this manual is for information purposes only. This manual is subject to change without notice.

# In-System Emulator Boards

## Table of Contents

Section	Description	Page
<b>Chapter 1. Introduction</b>		8-116
<b>Chapter 2. General Description of E02/E04 Boards</b>		
2.1	Physical Features .....	8-117
2.2	Jumpers .....	8-117
2.3	Turret Terminals .....	8-119
2.4	Reset Switch .....	8-119
2.5	Edge Connector Pin Assignments .....	8-119
<b>Chapter 3. Operating Considerations for E02/E04 Boards</b>		
3.1	Emulator Cables .....	8-122
3.2	Clock Timing .....	8-122
3.3	Single Supply Operation .....	8-122
3.4	COP400 Family Chips Emulation Requirements .....	8-122
<b>Chapter 4. Function Verification of the E02/E04L Boards and the E24 Board Using COP404</b>		8-124
<b>Chapter 5. Description of E24 Emulator Board</b>		
5.1	Physical Features .....	8-128
5.2	Jumpers .....	8-128
5.3	Turret Terminals .....	8-128
5.4	Reset Switch .....	8-129
5.5	Edge Connector Pin Assignments .....	8-129
<b>Chapter 6. Functional Verification of E24 Board Using COP2404</b>		8-131
<b>Figure</b>	<b>Illustrations</b>	<b>Page</b>
2-1	COP400-E02/E04L In-System Emulator Layout .....	8-118
3-1	Crystal Oscillator Component Carrier Schematic .....	8-122
3-2	LC Oscillator Schematic .....	8-123
3-3	COP402/404L to COP410L/411L RAM Mapping .....	8-123
5-1	COP400-E24 In-Circuit Emulator Layout .....	8-129
<b>Table</b>	<b>Tables</b>	<b>Page</b>
1-1	Emulator Boards and ROMless Parts for COP4XX Device Emulation .....	8-116
1-2	COP400 PDS Development System Software and Hardware .....	8-116
1-3	Starplex Development System Software and Hardware .....	8-116
2-1	Standard Jumper Configurations .....	8-120
2-2	Edge Connector Assignments .....	8-121
3-1	RC Oscillator Component Values .....	8-122
3-2	Crystal Oscillator Component Values .....	8-122
3-3	LC Oscillator Component Values .....	8-123
5-1	Standard Jumper Configuration for COP400-E24 Board .....	8-128
5-2	Edge Connector Assignments .....	8-130



# Introduction

The COP400 Emulator Board enables in-system emulation of the COP400 Microcontroller family. Table 1-1 shows the emulator boards and the ROMless microcontroller combinations required to emulate the various COP4XX devices.

The emulator board may be used stand-alone with EPROMs and external power supply, or as a peripheral to a development system. The COP400 Emulator Boards are designed to interface with either the COP400-PDS Development System using the target board and software revisions shown in Table 1-2, or the STARPLEX™ Development System, using the target board and software revisions shown in Table 1-3.

When used in conjunction with a development system, the emulator adds the capabilities of real-time program tracing, breakpoint/singlestepping, and speedy program updating, resulting in rapid program evolution from conception through debug to final product.

**Note**

*The user should read this manual thoroughly before attempting COP4XX Emulation.*

**Table 1-1. Emulator Boards and ROMless Parts For COP4XX Device Emulation**

ROMless Part	Emulator Board	Parts Emulated	Refer Chapters
COP401L†	COP400-E02 COP400-E04L	COP410L COP411L	2, 3, 4
COP402	COP400-E02	COP420 COP421 COP422	2, 3, 4
COP404L	COP400-E04L	COP420L COP421L COP422L COP444L COP445L	2, 3, 4
COP404	COP400-E24	COP440 COP441 COP442	4, 5
COP2404††	COP400-E24	COP2440 COP2441 COP2442	5, 6

†The COP401L has the CKO pin selected as the RAM Keep Alive option. This pin must be connected to the V<sub>CC</sub> power supply in the user's system and J6 installed on the COP400-E02 or COP400-E04L Board.

††As shipped, the E24 board contains a COP404 ROMless part. For emulating the COP 2440, 2441 or 2442, install the COP2404 shipped with the board in socket U2.

**Table 1-2. COP400-PDS Development System Software and Hardware**

ROMless Part	Emulator Board	Target Board	Software
COP401L†	COP400-E02/ COP400-E04L	980306552-A	COP400-D02 Rev A
COP402	COP400-E02	980305551-F	COP400-D01 Rev D
	COP400-E02	980306552-A	COP400-D02 Rev A
COP404L	COP400-E04L	980305551-F	COP400-D01 Rev D
	COP400-E04L	980306552-A	COP400-D02 Rev A
COP404	COP400-E24	980306552-A	COP400-D02 Rev A
COP2404	COP400-E24	980306552-A	COP400-D02 Rev A

†The COP401L has the CKO pin selected as the RAM Keep Alive option. This pin must be connected to the V<sub>CC</sub> power supply in the user's system and W4 installed on the COP400-E02 or COP400-E04L Board.

**Table 1-3. STARPLEX Development System Software and Hardware**

ROMless Part	Emulator Board	Target Board	Software
COP401L†	COP400-E02/ COP400-E04L	980306254-A	440306254-001 Rev A or B
COP402	COP400-E02	980306254-A	440306254-001 Rev A or B
COP404L	COP400-E04L	980306254-A	440306254-001 Rev A or B
COP404	COP400-E24	980306254-A	440306254-001 Rev B
COP2404	COP400-E24	980306254-A	440306254-001 Rev B

# General Description of E02/E04L Boards

## 2.1 Physical Features

The emulator is a double-sided printed circuit board mounted on four 0.5-inch nylon stand-offs. Figure 2-1 is a drawing of the board with its emulator cables removed. Processing is carried out by a ROMless microcontroller located top-center on the board. To the left of the ROMless device are four single in-line connectors and one 20-pin socket used in receptacles for the DIP-to-DIP emulator cables. The DIP-to-DIP cables connect the emulator board to the Target System. In the center of the board are four MM5204 PROM sockets. The PROM socket labeled "PROM 0" is for COP addresses 0-1FFH, the socket labeled "PROM 1" is for addresses 200-3FFH, the socket labeled "PROM 2" is for COP addresses 400-5FFH, and the socket labeled "PROM 3" is for COP addresses 600-7FFH. Below and to the right of these PROM sockets is a socket for an MM2716 PROM. Below the PROM sockets at the bottom of the board is a 50-pin edge connector used to interface to the development system via the emulator board cable. Pin 1 of this cable should match up with pin 1 of the edge connector which is shown in the lower right-hand corner of Figure 2-1.

### WARNING

*Never connect or disconnect the emulator board from the emulator board cable while the development system is turned on; permanent development system and/or emulator damage may result.*

## 2.2 Jumpers

The emulator board has wire-wrap pin jumpers. J1, J3, J4, J5, J6 and J7 are located to the right of the ROMless microcontroller, J2 is in the upper left-hand corner of the board, J8, J9, J10, and J11 are located in the lower right-hand corner of the board. See Table 2-1 for the standard jumper configurations.

### 2.2.1 J1

J1 is a set of seven jumpers. However, only three are connected at any one time. J1 is used to select the signal assignments of pins 13, 14 and 15 on the 20-pin COP411L emulator cable socket. For 411L operation J1-1, J1-4, and J1-6 should be jumpered. To emulate 20-pin COP400 devices not discussed in this manual, contact the factory for the correct J1 configuration.

### 2.2.2 J2

J2 jumpers the +5 volt power bus on the emulator board to the  $V_{CC}$  of the ROMless microcontroller and emulator cables.

### CAUTION

*The user must not connect the target system power supply to the development system supply via the 4XX emulation cables. This could destroy one or both supplies.*

J2 should be removed if the board is connected to the development system and the user's system power is connected to the 4XX emulator cables. On the other hand, if the board is being used stand-alone with external power supplies, J2 may be left in place. The target system's power should be adequately bypassed and regulated to eliminate random malfunctioning of the ROMless microcontroller due to power glitches.

### 2.2.3 J3

The emulator board is supplied with an RC oscillator. J3 jumpers the output of this oscillator to the CKI input of the ROMless microcontroller. J3 should be removed if the user plans to generate a clock signal external to the board. Section 3.2 contains more information concerning J3 and clock timing.

### 2.2.4 J4

J4 connects CKO of the ROMless microcontroller to pin 11 of U7. J4 should be jumpered if U7 is replaced with any of the component carriers described in Section 3.2 of this manual.

### 2.2.5 J5

Jumper J5 connects CKI of the ROMless microcontroller to the emulator cable sockets. This jumper is installed only when the clock is being furnished from the target system.

### 2.2.6 J6

Jumper J6 connects CKO of the ROMless microcontroller to the emulator cable sockets. When using COP402 or COP404L, this jumper is installed only when the target system will ultimately use the clock from the user's COP4XX device. Due to cable capacitance, care should be exercised when using this jumper. When using a COP401L, this jumper must be installed and the user should tie CKO to  $V_{CC}$  in his system.

### 2.2.7 J7

Jumper J7 controls the RC oscillator frequency. With J7 removed, the frequency is 3.5 MHz. With J7 installed, the frequency is 1.7 MHz. Jumper J7 should be installed for emulation of COP4XXL devices.

### 2.2.8 J8

Jumper J8 determines the pull-up load resistor value and power supply for the reset line to the ROMless microcontroller. The COP404L has the power fail reset option implemented. With J8 in the "A" position, a 510 $\Omega$  pull-up to 5V is installed. This allows the ROMless device to operate normally and is the preferred position for J8. With J8 in the "B" position, the reset line is pulled up to the chip  $V_{CC}$  by a 20k $\Omega$  resistor. This position allows the internal power fail reset on the COP404L to function properly.

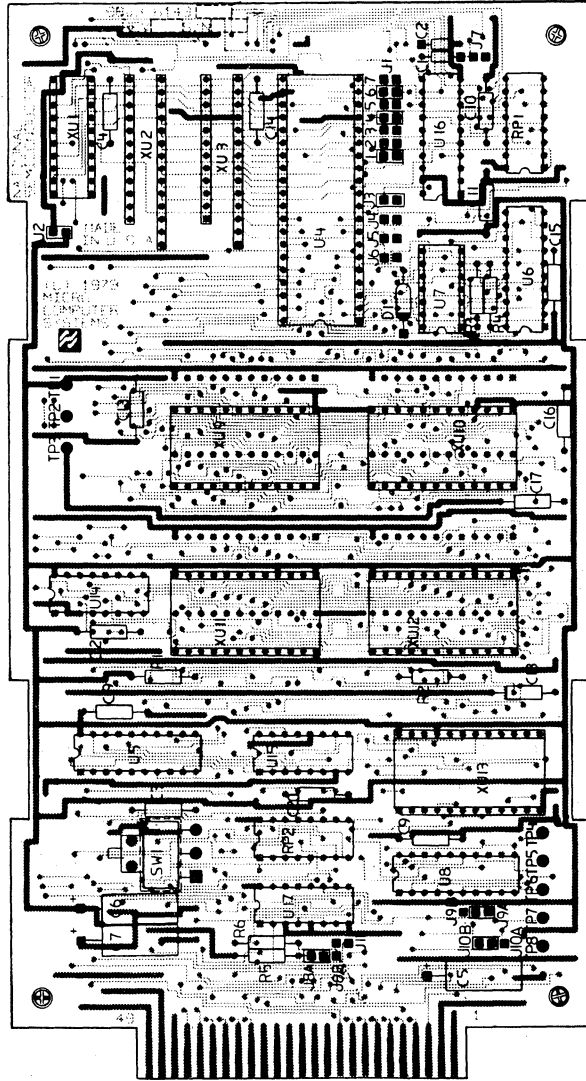


Figure 2-1. COP4000-E02/E04L In-System Emulator™ Layout

### 2.2.9 J9

J9 configures address line A10 for the various devices being emulated. In the "A" position, A10 is an address line, necessary for emulation of the COP444L/445L devices. In the "B" position, A10 is held in the low state, necessary for emulation of all other devices.

### 2.2.10 J10

Because SKIP/P10 are multiplexed on one pin, emulation of the COP404L device requires J10 in the "A" position. When emulating the COP401L and the COP402, place J10 in position "B."

### 2.2.11 J11

J11 generates the special clock used for the COP404L device and is installed only when this device is used.

## 2.3 Turret Terminals

The board contains eight turret-type terminals suitable for temporary connections via alligator clips, Q-balls, etc. Three of these terminals are used for power, four as logic inputs to the development system and one as a logic output of the development system.

### 2.3.1 Emulator Power Terminals

Two power supplies (+5 and  $-12V_{DC}$ ) are necessary to operate the board stand-alone with MM5204 EPROMs. These voltage inputs and their returns are supplied to the board via the three terminals located on the left edge of the board marked  $V_{CC}$ ,  $-12V$ , and GND. THESE POSTS DO NOT ALLOW THE USER ACCESS TO DEVELOPMENT SYSTEM SUPPLIES; they supply power to the emulator board used independently of the development system. Typical power consumption of the board with four EPROMs is 250mA for +5V and 60mA for the  $-12V$  input. For single +5V operation using DM74S474 bipolar PROMs (see Section 3.3), the +5V current drain is approximately 500mA.

### 2.3.2 External Event Terminals

Four external event terminals (EX1-EX4) are located on the right side of the board. The logical inputs (TTL levels) on these high impedance pins are stored in

\*Denotes an active low signal.

TRACE memory along with the COP4XX program counter values and the skip line status during a TRACE operation. Transitions on EX1 and EX2 may be used to initiate TRACE or BREAKPOINT operations. For more information concerning the external event terminals, consult Chapters 2 and 9 of the COP400 Product Development System User's Manual, or Chapter 4 of the STARPLEX™ SPM-A15 Operator's Manual.

### 2.3.3 Trigger Out

Trigger out (TO) is located beneath EX1-EX4 on the emulator board. TO is an open-collector development system output that makes a positive transition each time a TRACE or BREAKPOINT is initiated. In certain applications TO is useful for triggering oscilloscopes or logic analyzers. (Note: When using a PDS target board, Part No. 980305551, Rev. F, or earlier, TO will continue to make positive transitions every 256 trigger conditions following the actual TRACE or BREAKPOINT.)

### 2.4 Reset Switch

The reset switch is located in the lower left corner of the emulator. When pressed, the reset switch clears the COP4XX program counter, registers and outputs. The COP4XX will remain in this reset condition until the switch is released. Pressing this switch also causes the RESET\* pin (open-collector output, resistor pull-up to  $V_{CC}$ ) on the emulator cable sockets to go low. This switch is for stand-alone operation of the emulator board. When emulating using the development system program COPMON, it is preferable to use the 'R' (reset) command.

## 2.5 Edge Connector Pin Assignments

The 50-pin edge connector located at the bottom of the board provides interface to the development system (refer to Chapter 2 of the PDS User's Manual or Chapter 3 of STARPLEX SPM-A15 Operator's Manual). Table 2-2 contains the names and a brief description of each signal.

Table 2-1. Standard Jumper Configurations

Jumper	410L/411L	420L/421L	444L/445L	420/421	See Section
J1-1	IN	N/A	N/A	N/A	
J1-2	OUT	N/A	N/A	N/A	
J1-3	OUT	N/A	N/A	N/A	
J1-4	IN	N/A	N/A	N/A	
J1-5	OUT	N/A	N/A	N/A	
J1-6	IN	N/A	N/A	N/A	
J1-7	OUT	N/A	N/A	N/A	
J2	IN	IN	IN	IN	
J3	IN	IN	IN	IN	
J4	OUT	OUT	OUT	OUT	
J5	OUT	OUT	OUT	OUT	
J6	Note 2	OUT	OUT	OUT	2.2.6
J7	IN	IN	IN	OUT	
J8-A	IN	IN	IN	IN	
J8-B	OUT	OUT	OUT	OUT	
J9-A	OUT	OUT	IN	OUT	
J9-B	IN	IN	OUT	IN	
J10-A	OUT	IN	IN	OUT	2.2.10
J10-B	IN	OUT	OUT	IN	2.2.10
J11	Note 1	IN	IN	OUT	2.2.11

**Note 1:** This jumper must be in when using the COP404L ROMless Microcontroller and out when using the COP402 or COP401L ROMless Microcontrollers.

**Note 2:** When using a COP401L ROMless part, this jumper must be installed and CKO tied to  $V_{CC}$  in the user system. For COP402 or COP404L operation this jumper should be out.

Table 2-2. Edge Connector Assignments

Connector No.	Name	Description
1	GND	Signal and power return
2	GND	Signal and power return
3	V <sub>CC</sub>	+5V <sub>DC</sub> power from development system
4	V <sub>CC</sub>	+5V <sub>DC</sub> power from development system
5	EX2	Buffered external event
6	EX1	Buffered external event
7	EX4	Buffered external event
8	EX3	Buffered external event
9	CLK	Buffered AD/DATA* signal from COP4XX
10	SKIP	COP4XX skip status line
11	A8	COP4XX program counter address bit
12	A9	Address bit
13	A3	Address bit
14	A7	Address bit
15	A1	Address bit
16	A2	Address bit
17	A4	Address bit
18	A0	Least significant address bit
19	A6	Address bit
20	A5	Address bit
21	Not used	
22	A10	Most significant address bit
23	Not used	
24	Not used	
25	Not used	
26	Not used	
27	Not used	
28	Not used	
29	Not used	
30	Not used	
31	Not used	
32	Not used	
33	B0	Least significant COP object code bit
34	B7	Most significant COP object code bit
35	B2	Object code bit
36	B5	Object code bit
37	B3	Object code bit
38	B4	Object code bit
39	B6	Object code bit
40	B1	Object code bit
41	TRIGGER OUT	BREAKPOINT/TRACE indicator
42	Not used	
43	RST*	Same as RESET*
44	PROM DISABLE*	Select PROM or Shared Memory mode
45	See Note 1	
46	See Note 1	
47	V <sub>CC</sub>	+5V <sub>DC</sub> power from development system
48	V <sub>CC</sub>	+5V <sub>DC</sub> power from development system
49	GND	Power and signal return
50	GND	Power and signal return

**Note 1:** Pins 45 and 46 are used as follows:

PDS	with target board 980306552 REV A or later, normally not used. with target board 980305551 REV F or earlier, -12V <sub>DC</sub> from the PDS.
STARPLEX	with target board 980306254, normally not used. However, jumper W5 on the target board may be installed to supply -12V <sub>DC</sub> to the emulator board.

# Operating Considerations For E02/E04L Boards

## 3.1 Emulator Cables

Three DIP-to-DIP cables (20-, 24-, and 28-pin) are supplied with the board. The user should note the orientation of pin 1 on the three emulator cable sockets (see Figure 2-1). The 20-pin socket (in the upper left corner of the emulator) is used for emulating a COP411L device. Pin 1 of the device cable should be oriented away from the center of the board. The four SIP (single-in-line package) sockets to the right of the 20-pin DIP (dual-in-line package) socket are configured as either a 24- or a 28-pin socket. Pin 1 for both of these sockets is oriented toward the center of the board. The 24-pin socket is used to emulate a 410L, 421L, 421, or 445L device. The 28-pin socket is used to emulate the 420L, 420, and 444L devices.

Note: Only one COP400 family device may be emulated at any one time.

## 3.2 Clock Timing

### 3.2.1 RC Oscillator

The emulator has an on-board 3.5/1.7 MHz RC oscillator. Jumper J7 controls the frequency — with J7 removed it is 3.5 MHz, with J7 installed it is 1.7 MHz. Lower frequencies can be obtained by performing the following steps:

1. Replace the 74LS14 of U7 with a 74C14.
2. Remove J7.
3. Remove R4.
4. Set R3 and C1 to the values in Table 3-1.

### 3.2.2 Crystal Oscillator

The COP402 on the COP400-E02 board has a crystal oscillator option enabling the user to emulate with a crystal-controlled clock. Due to emulator cable capacitance and inductance, the use of the crystal oscillator on the user's target prototype system requires three changes to the circuit: first, replace the 74LS14 (U7) with a 14-pin component carrier containing the circuit shown in Figure 3-1; second, install J3 and J4; and third, remove J5 and J6. Table 3-2 contains the various values of Rx1, Rx2, and Cx needed for three standard crystal frequencies.

### 3.2.3 LC Oscillator (COP402 only)

Use of the LC oscillator requires replacing U7 with a 14-pin carrier containing components for an LC oscillator. Figure 3-2 shows the schematic and Table 3-3 contains sample values for a COP420 LC oscillator. Jumpers J3 and J4 must be installed and jumpers J5 and J6 must be removed.

## 3.3 Single Supply Operation

The board is factory equipped with sockets for MM5204 EPROMs which require multiple power supplies. For single supply (+5V) operation, a socket for an MM2716 EPROM can be used instead of MM5204 EPROMs. The MM2716 must be placed in the socket

U13 in the lower right of the board. Also, the board can be modified to accept DM74S474 or DM74S475 bipolar PROMs. The mounting holes for these PROMs are located above the EPROM sockets on Figure 2-1 and are labeled S474. Pin 1 is marked.

## 3.4 COP400 Family Chips Emulation Requirements

As shown in Table 1-1, emulation of a particular COP400 series device is normally done with the corresponding emulator board. For example, a COP400-E04L board is required to emulate a COP444L. In some cases, it is possible to emulate a COP400 device on a board not specifically designed for it, provided certain precautions are observed.

The COP400-E04L board, which contains a COP404L, can be used to emulate any COP400L device. The COP400-E02 board, which contains a COP402, can be used to emulate any COP400 device except the COP444L and the COP445L.

There may be substantial differences between the chip being emulated and the actual device on which the emulation is done. It is essential that all of the following points are observed before emulation.

Table 3-1. RC Oscillator Component Values

R3 ( $\Omega$ )	C1 (pF)	Oscillator Frequency (MHz)	Instruction Cycle ( $\mu$ s)	
			COP402 ( $\div 16$ )	COP404L ( $\div 32$ )
1.2k	100	1.0	16	32
4.7k	100	0.5	32	64

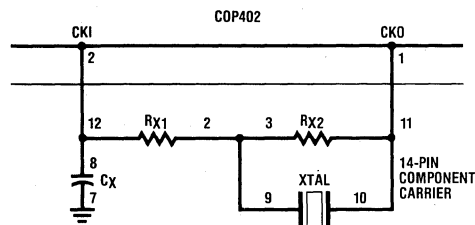


Figure 3-1. Crystal Oscillator Component Carrier Schematic

Table 3-2. Crystal Oscillator Component Values

Rx1 ( $\Omega$ )	Rx2 ( $\Omega$ )	Cx (pF)	XTAL Frequency	
			Frequency (MHz)	Instruction Cycle ( $\mu$ s)
1.0k	1.0M	27	4.00	4.0
1.0k	1.0M	27	3.58	4.5
1.0k	1.0M	56	2.09	7.7

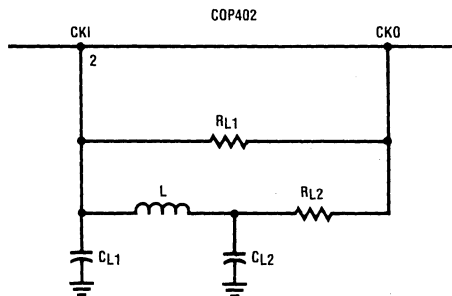


Figure 3-2. LC Oscillator Schematic

Table 3-3. LC Oscillator Component Values

RL1 (Ω)	RL2 (Ω)	CL1 (pF)	CL2 (pF)	L (μH)	Oscillator Frequency (MHz)	Instruction Cycle (μs)
10.0M	510	100	100	22	4.0	4.0
10.0M	510	500	100	22	3.0	5.3
10.0M	510	200	500	22	2.6	6.2
10.0M	510	200	25000	22	1.4	11.4

### 3.4.1 RAM Registers

The following table indicates how much RAM is available on a given COP400 microcontroller.

Device	RAM
401L, 410L, 411L	4 registers × 8 digits
402, 420, 420L, 421, 421L	4 registers × 16 digits
404L, 444L, 445L	8 registers × 16 digits

For example, a COP410L can be emulated using a COP404L. The 404L has eight 16-digit registers and is emulating with more RAM than is available in the final device. Figure 3-3 illustrates how a COP404L RAM register maps into a COP410L register of eight digits. The 401L has no 2-byte LBI instructions, only single-byte instructions of LBI R,0 and LBI R,9 through LBI R,15 (where R = 0, 1, 2, 3). Nevertheless, the Bd register is still four bits wide, and instructions such as XDS and XIS will still generate a skip at their respective 0-15 and 15-0 Bd value boundaries. The 410L has a 2-level stack, compared to the 404L's 3-level stack.

The COP PDS Assembler (Rev. B) will flag all RAM reference and stack reference instructions with an asterisk if the .CHIP directive specifies a COP410L or COP411L.

The user should study these instructions in their program contents and verify that they are operating correctly.

### 3.4.2 Program Memory (ROM)

The following table indicates the maximum ROM address space available on COP400 microcontrollers.

Device	ROM
401L, 410L, 411L	512 bytes
402, 420, 420L, 421, 421L	1024 bytes
404L, 444L, 445L	2048 bytes

For example, if emulating 410L using a 402, the program size must be restricted to a maximum of 512 bytes.

In all cases, the program size will be properly restricted if the correct .CHIP directive is used in the assembly language source code.

### 3.4.3 Electrical Characteristics

The user must be aware of any differences in electrical characteristics between the emulating chip and the final device. A COP402, for example, can operate at a higher clock rate than a COP421L. Different chips may have different drive capabilities, and user options may cause the functions of some pins on the final device to be different from the functions of those pins on the emulating chip.

If there are questions, call COPST<sup>™</sup> Applications at (408) 721-5582.

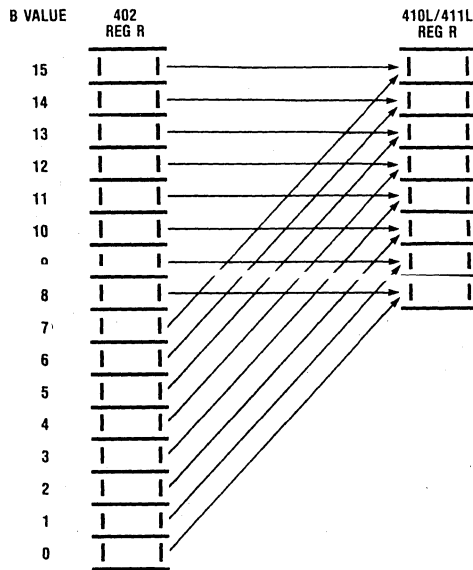


Figure 3-3. COP402/404L to COP410/411L RAM Mapping



# Functional Verification of the E02/E04L Boards and the E24 Board Using COP404

Due to the board's physical location, external to the development system, it is easily damaged. Always observe the following:

1. User power supplies are adequately bypassed and supplying the correct voltage.
2. Development system power is not connected to user power.
3. Cables are correctly installed.
4. Device input/output ratings are not exceeded.
5. PROMs are in their correct sockets and properly oriented.
6. The COP4XX is receiving a valid clock signal.

If a mishap or malfunction does occur, National Semiconductor's Microcomputer Technical Support Manager can be contacted at (408) 721-6803. Questions concerning actual operation of the board or customer use of a COP400 device should be referred to the COPSTM Application Group at (408) 721-5582.

Alternatively, if the board develops a problem and circumstances do not allow sufficient time to send it back to National, there is a series of COPMON commands that may be used to isolate faulty component(s). Before attempting the following diagnostic aids, the user should study Section 2.5 of this manual and the schematic supplied with the emulator board. The user will also need a functional development system and an emulator board cable.

PDS users should study Chapter 9 of the PDS User's Manual.

STARPLEX™ users should consult the SPM-A15 Operator's Manual (Manual No. 420306254) for information on COPMON.

1. With power turned off, connect the board to the Development System, making sure all jumpers are correctly assigned. Correct jumper assignments are determined by the ROMless microcontroller on the board. Refer to Section 2.2 for E02/E04L boards and Section 5.2 for the E24 board.
2. Turn power on and load COPMON. When prompted, specify chip number 444 if verifying an E02 or an E04L emulator board; specify chip 440 if using an E24 board with a COP404 ROMless microcontroller.

Note: Underline indicates user inputs.

PDS Example:

```

CR
EXEC, REV:A
X> @COPMON
COPMON, REV: D
CHIP NUMBER (DEFAULT = 420)? 444 (or 440)

```

STARPLEX Example:

```

>COPMON
COPMON, REV: B, <date>
CHIP NUMBER (DEFAULT = 420)? 444 (or 440)
C>

```

Note: Specifying the CHIP NUMBER as 444 or 440 will allow the emulator to access all of shared memory.

3. Load shared memory with CLRA (object code = 00H) instructions.

```
C> DE 0, 0/L
```

4. Specify and perform a TRACE IMMEDIATE

```

C> TR I
TRACE ENABLED:
IMED OCCUR: 1 PRIOR: 0 GO: N
C> G

```

COPMON should come back with the following message:

```
TRACED ON IMED AT A:000
```

If it does not, then the CLK signal described in Section 2.5 is not being generated by the COP4XX and/or the CLK signal is not reaching the system. Probable faulty circuits:

E02/E04L board	E24 with COP404
COP 4XX (U4)	COP404 (U2)
81LS95 (U8)	74LS14 (U1)

The user should also verify that the COP4XX is receiving a valid clock input.

If COPMON executed the TRACE properly, the COP program counter can now be examined with the TYPE command.

```
C> TY
```

0	0 A:000	E:1111
1	1 A:000	E:1111
2	2 A:000	E:1111
3	3 A:000	E:1111
4	4 A:000	E:1111
5	5 A:000	E:1111
6	6 A:000	E:1111
7	7 A:000	E:1111
8	8 A:000	E:1111
9	9 A:000	E:1111
10	10 A:000	E:1111
11	11 A:000	E:1111
12	12 A:000	E:1111
13	13 A:000	E:1111
14	14 A:000	E:1111
15	15 A:000	E:1111

Note all the address values (A:XXX) shown are zero. This is correct because the TRACE operation was begun before COPMON let RST\*/RESET\* go to a high level. If one or more of the program counter bits are stuck high, the TYPE command might yield the following information:

C>TY

0	0A:009	E:1111
1	1A:009	E:1111
2	2A:009	E:1111
3	3A:009	E:1111
4	4A:009	E:1111
5	5A:009	E:1111
6	6A:009	E:1111
7	7A:009	E:1111
8	8A:009	E:1111
9	9A:009	E:1111
10	10A:009	E:1111
11	11A:009	E:1111
12	12A:009	E:1111
13	13A:009	E:1111
14	14A:009	E:1111

With this information, the user can generally isolate which address line (A0-A10) is malfunctioning. Probable faulty circuits:

E02/E04L board	E24 with COP404
81LS95 (U8)	81LS95 (U10)
81LS95 (U16)	81LS95 (U11)
74LS373/74C373 (U6)	74LS374 (U7)
	74LS374 (U9)

5. Enter another TRACE IMMEDIATE command to test RST\*/RESET\* and proper binary operation of the address lines.

C>G

TRACED ON IMED AT A:01B

C>TY

0	0A:01B	E:1111
1	1A:01C	E:1111
2	2A:01D	E:1111
3	3A:01E	E:1111
4	4A:01F	E:1111
5	5A:020	E:1111
6	6A:021	E:1111
7	7A:022	E:1111
8	8A:023	E:1111
9	9A:024	E:1111
10	10A:025	E:1111
11	11A:026	E:1111
12	12A:027	E:1111
13	13A:028	E:1111
14	14A:029	E:1111
15	15A:02A	E:1111

C>TY

16	16A:02B	E:1111
17	17A:02C	E:1111
18	18A:02D	E:1111
19	19A:02E	E:1111
20	20A:02F	E:1111
21	21A:030	E:1111
22	22A:031	E:1111
23	23A:032	E:1111
24	24A:033	E:1111
25	25A:034	E:1111
26	26A:035	E:1111
27	27A:036	E:1111
28	28A:037	E:1111
29	29A:038	E:1111
30	30A:039	E:1111
31	31A:03A	E:1111

If the RST\*/RESET\* line is stuck low, the addresses shown above would have remained at zero. Probable faulty circuits:

E02/E04L board	E24 with COP404
COP4XX (U4)	COP404 (U2)
Dev. System	Dev. System

The COP addresses from this second TRACE IMMEDIATE operation should be inspected for monotonically increasing binary values from 0 to the highest ROM address and wraparound from this address to 0. (The highest address depends on the ROMless microcontroller being used and is 1FF for a COP401L, 3FF for a COP402 and 7FF for a COP404L or COP404.) This can be done by additional TRACE and TYPE commands.

If several address lines are shorted or inoperative, a TYPE command might yield program counter values like the following:

C>TY

0	0A:38B	E:1111
1	1A:38B	E:1111
2	2A:38D	E:1111
3	3A:38D	E:1111
4	4A:38F	E:1111
5	5A:38F	E:1111
6	6A:399	E:1111
7	7A:399	E:1111
8	8A:39B	E:1111
9	9A:39B	E:1111
10	10A:39D	E:1111
11	11A:39D	E:1111
12	12A:39F	E:1111
13	13A:39F	E:1111
14	14A:399	E:1111
15	15A:399	E:1111

```
C>TY
16 16 A:39B E:1111
17 17 A:39B E:1111
18 18 A:39D E:1111
19 19 A:39D E:1111
20 20 A:39F E:1111
21 21 A:39F E:1111
22 22 A:3A9 E:1111
23 23 A:3A9 E:1111
24 24 A:3AB E:1111
25 25 A:3AB E:1111
26 26 A:3AD E:1111
27 27 A:3AD E:1111
28 28 A:3AF E:1111
29 29 A:3AF E:1111
30 30 A:3A9 E:1111
31 31 A:3A9 E:1111
```

Probable faulty circuits:

<b>E02/E04L board</b>	<b>E24 with COP404</b>
81LS95 (U16)	81LS95 (U10)
81LS95 (U8)	81LS95 (U11)
74LS373/74C373 (U6)	74LS374 (U7)
	74LS374 (U9)

6. Given 1 A:001 peration of the board to this point,  
 test ti 2 A:002 program data bits (B0-B7) by insert-  
 ing va 3 A:003 p commands into memory and using  
 TRAC 4 A:004 / that the jump occurred. Probable  
 faulty circuit is:

<b>E02/E04L board</b>	<b>E24 with COP404</b>
81LS95 (U5)	81LS95 (U6)

Insert a jump to location 0 at address 3E and set up,  
 execute, and list the trace.

```
C>PU 3E, JP 0
C>TR 3E, 1, 0
TRACE ENABLED:
A:03E OCCUR: 1 PRIOR: 0 GO: N
```

```
C>G
TRACE ON A:03E AT A:03E
C>TY 0/4
```

```
0 0 A:03E E:1111
1 1 A:000 E:1111
2 2 A:001 E:1111
3 3 A:002 E:1111
4 4 A:003 E:1111
```

If the second TRACE memory location does not  
 contain A:000, then the board is not recognizing the  
 JP 0 instruction (object code = C0H) properly. If the  
 JP 0 is working, then high levels on B6 and B7 from  
 the PDS are being recognized by the board. Proper  
 high levels on B0-B5 may now be tested by succes-  
 sively replacing the JP 0 instruction with the follow-  
 ing jumps: JP 1 (C1H), JP 2 (C2H), JP 4 (C4H), JP 8  
 (C8H), JP 10 (D0H), and JP 20 (E0H). The necessary  
 COPMON commands are:

```
C>PU 3E, JP 1
C>G
TRACED ON A:03E AT A:03E
C>TY 0/4
0 0 A:03E E:1111
1 1 A:001 E:1111
2 2 A:002 E:1111
3 3 A:003 E:1111
4 4 A:004 E:1111
```

Note that the contents of trace location 1 should  
 be A:001. If B0 and B1 were shorted or inoperative,  
 the TYPE command might yield the following  
 information:

```
C>G
TRACED ON A:03E AT A:03E
C>TY
```

```
0 0 A:03E E:1111
1 1 A:003 E:1111
2 2 A:004 E:1111
3 3 A:005 E:1111
4 4 A:006 E:1111
5 5 A:007 E:1111
6 6 A:008 E:1111
7 7 A:009 E:1111
8 8 A:00A E:1111
9 9 A:00B E:1111
10 10 A:00C E:1111
11 11 A:00D E:1111
12 12 A:00E E:1111
13 13 A:00F E:1111
14 14 A:010 E:1111
15 15 A:011 E:1111
```

```
Continuing the test:
C>TR 3E, 1, 0
TRACE ENABLED:
A:03E OCCUR: 1 PRIOR: 0 GO: N
```

```
C>PU 3E, JP 2
TRACED ON A:03E AT A:03E
C>TY 0/4
0 0 A:03E E:1111
1 1 A:002 E:1111
2 2 A:003 E:1111
3 3 A:004 E:1111
4 4 A:005 E:1111
```

```
C>PU 3E, JP 4
C>G
TRACED ON A:03E AT A:03E
C>TY 0/4
```

```

0      0 A:03E      E:1111
1      1 A:004      E:1111
2      2 A:005      E:1111
3      3 A:006      E:1111
4      4 A:007      E:1111

```

C>PU 3E, JP 8

C>G

TRACED ON A:03E AT A:03E

C>TY 0/4

```

0      0 A:03E      E:1111
1      1 A:008      E:1111
2      2 A:009      E:1111
3      3 A:00A      E:1111
4      4 A:00B      E:1111

```

C>PU 3E, JP 10

C>G

TRACED ON A:03E AT A:03E

C>TY 0/4

```

0      0 A:03E      E:1111
1      1 A:010      E:1111
2      2 A:011      E:1111
3      3 A:012      E:1111
4      4 A:013      E:1111

```

C>PU 3E, JP 20

C>G

TRACED ON A:03E AT A:03E

C>TY 0/4

```

0      0 A:03E      E:1111
1      1 A:020      E:1111
2      2 A:021      E:1111
3      3 A:022      E:1111
4      4 A:023      E:1111

```

The final test ensures that B6 and B7 are not shorted together and that the SKIP line is functioning. Test this with a 2-byte JMP 1 instruction (Op code = 6001H).

C>PU 3E, JMP 1

C>G

TRACED ON A:03E AT A:03E

C>TY 0/4

```

0      0 A:03E      E:1111
1      1 A:03F SKIP E:1111
2      2 A:001      E:1111
3      3 A:002      E:1111
4      4 A:003      E:1111

```

If trace location 2 does not contain A:001, then the JMP 1 instruction was not recognized. Also, if the jump to 1 was made, but location 1 does not show a SKIP, there is a problem with the SKIP line. Check:

<b>E02/E04L board</b>	<b>E24 with COP404</b>
COP4XX (U4)	COP404 (U2)
Dev. System	Dev. System

If the emulator passes all the above tests, the supporting circuitry to the COP4XX is functional. It is imperative that all these tests be performed with a completely operational development system. A malfunctioning emulator board is difficult to discern from a malfunctioning system. If the user still experiences difficulties during program emulation, the Microcomputer Technical Support Manager, (408) 721-6803, should be contacted.

# Description of E24 Emulator Board

## 5.1 Physical Features

The emulator is a double-sided printed circuit board mounted on four 0.5-inch stand-offs. Figure 5-1 contains a drawing of the board with its emulator cables removed. Processing is carried out by a ROMless microcontroller located top-left center on the board. To the right of the ROMless microcontroller are connectors used as receptacles for the DIP-to-DIP emulator cables. The DIP-to-DIP cables connect the emulator board to the target system. In the center of the board is a PROM socket, at the bottom of the board is a 50-pin edge connector used to interface to the development system via the emulator board cable. Pin 1 of this cable should match up with pin 1 of the edge connector which is located in the lower right-hand corner of Figure 5-1. Pin 1 for all sockets is located on the end facing the 50-pin edge connector.

### Warning

*Never connect or disconnect the emulator board from the emulator board cable while the development system is turned on; permanent development system and/or emulator damage may result.*

## 5.2 Jumpers

The emulator board has wire-wrap pin jumpers. W1, W2, W3, W4, W5, W6, and W7 are located to the left of the ROMless microcontroller. W8 and W9 are located in the lower center of the board. W4 is a connection between the right-hand parts of W3 and W5. See Table 5-1 for the standard jumper configurations.

### 5.2.1 W1

The emulator board has an on-board 3.5 MHz RC oscillator for user emulation convenience. W1 jumpers the output of the oscillator into the CKI input of the ROMless part. W1 should be removed if the user plans to generate a clock signal external to the board.

### 5.2.2 W2

Jumper W2 connects CKI of the ROMless device to the emulator cable sockets. This jumper is installed only when the clock is being furnished from the user system.

### 5.2.3 W3

If the user selects option 21 equal to 2 or 3, CKO as a general purpose input, W3 is used to connect CKOI to the CKO pin of the emulator cable.

**Table 5-1. Standard Jumper Configuration For COP400-E24 Board**

W1 .....	IN
W2 .....	OUT
W3 .....	IN
W4 .....	OUT
W5 .....	IN
W6 .....	OUT
W7 .....	IN
W8 .....	OUT
W9 .....	IN

### 5.2.4 W4

If the user selects option 21 equal to 1, CKO as RAM Keep Alive, W4 is used to power CKO from the user's system. W4 is installed by connecting the right side of W3 to the right side of W5. See W5 and W7.

Note: In order for the emulator board to function, the RAM Keep Alive pin must be powered. See W4, W5, and W7. CARE MUST BE TAKEN TO ENSURE THAT THE RAM KEEP ALIVE VOLTAGE IS WITHIN  $V_{CC} \pm 1V$ .

### 5.2.5 W5

W5 is installed to power the CKO RAM Keep Alive from the development system power supply. See W4 and W7.

### 5.2.6 W6

W6 is installed to enable the emulation of the MICRO-BUS™ option. (Option 41 = 1)

### 5.2.7 W7

W7 jumpers the +5V power bus on the emulator board to the  $V_{CC}$  of the ROMless microcontroller and emulator cables.

### Caution

**THE USER MUST NOT CONNECT THE SYSTEM POWER SUPPLY TO THE DEVELOPMENT SYSTEM SUPPLY VIA THE 4XX EMULATION CABLES.**  
*This could destroy one or both supplies.*

W7 should be removed if the board is connected to the development system and the target system power is connected to the 4XX emulator cables. If the board is being used stand-alone with external power supplies, W7 may be left in place with no harmful effect. It should be noted that the target system's power should be adequately bypassed to eliminate random malfunctioning of the ROMless microcontroller due to power glitches.

### 5.2.8 W8

Jumper W8 is installed when an MM2724A EPROM is used in socket U8.

### 5.2.9 W9

Jumper W9 is installed when MM2716 or MM2724B EPROMs are used in socket U8.

## 5.3 Turret Terminals

The board contains seven turret-type terminals suitable for temporary connections via alligator clips, Q-balls, etc. Two of these terminals are used for power, four are used as logic inputs to the development system, and the last is a logic output of the development system.

### 5.3.1 Emulator Power Terminals

A +5V power supply is needed to operate the board stand-alone with an MM2716 EPROM. The voltage input and its return can be supplied to the board via the two terminals located on the left edge of the board marked  $V_C$  and GND. THESE POSTS ARE NOT MEANT TO

ALLOW THE USER ACCESS TO DEVELOPMENT SYSTEM SUPPLIES. They are to be used for supplying power to the emulator board when it is being used independently of the development system. Typical power consumption for the board using an MM2716 is 250 mA.

### 5.3.2 External Event Terminals

Four external event terminals (EX1-EX4) are located on the right side of the board. The logical inputs (TTL levels) on these high impedance pins are stored in TRACE memory along with COP4XX program counter values and the skip line status during a TRACE operation. In addition, transitions on EX1 and EX2 may be used to initiate TRACE or BREAKPOINT operations. For more information concerning the external event terminals, consult the Development System User's Manual.

### 5.3.3 Trigger Out

Trigger out (TO) is located directly beneath EX1-EX4 on the emulator board. TO is an open-collector development system output that makes a positive transition each time a TRACE or BREAKPOINT is initiated. TO can be used for triggering oscilloscopes or logic analyzers.

### 5.4 Reset Switch

The reset switch is located in the lower left corner of the emulator. When pressed, the reset switch clears the COP4XX program counter, registers and outputs. The COP4XX will remain in this reset condition until the switch is released. Pressing this switch will also cause the RESET\* pin (open-collector output, resistor pull-on to  $V_{CC}$ ) on the emulator cable sockets to go low. This switch is for stand-alone operation of the emulator board. When emulating using the development system program COPMON, it is preferable to use the 'R' (RESET) command.

### 5.5 Edge Connector Pin Assignments

The 50-pin edge connector located at the bottom of the board provides an interface to the development system. (Refer to Chapter 2 of the PDS User's Manual or Chapter 3 of the STARPLEX™ SPM-A15 Operator's Manual.) Table 5-2 contains the names and a brief description of each signal.

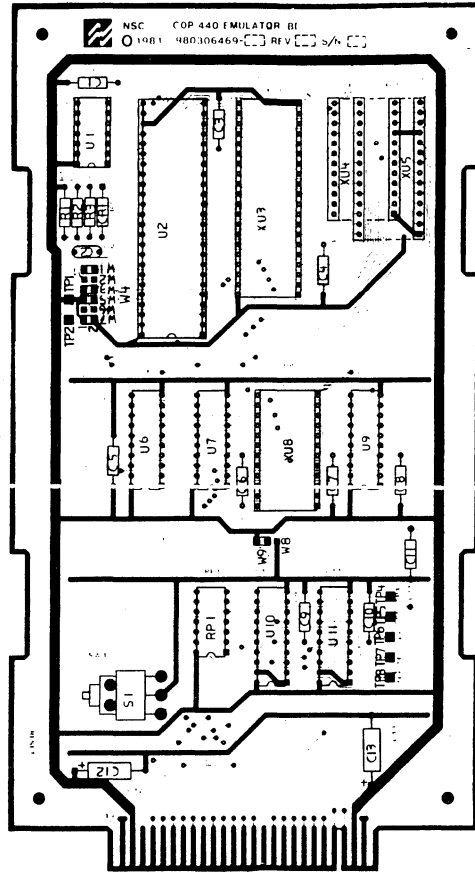


Figure 5-1. COP400-E24 In-Circuit Emulator Layout

\*Denotes an active low signal.

Table 5-2. Edge Connector Assignments

Connector No.	Name	Description
1	GND	Signal and power return
2	GND	Signal and power return
3	V <sub>CC</sub>	+5V <sub>DC</sub> power from development system
4	V <sub>CC</sub>	+5V <sub>DC</sub> power from development system
5	EX2	Buffered external event
6	EX1	Buffered external event
7	EX4	Buffered external event
8	EX3	Buffered external event
9	CLK	Buffered AD/DATA* signal from COP4XX
10	SKIP	COP4XX skip status line
11	A8	COP4XX program counter address bit
12	A9	Address bit
13	A3	Address bit
14	A7	Address bit
15	A1	Address bit
16	A2	Address bit
17	A4	Address bit
18	A0	Least significant address bit
19	A6	Address bit
20	A5	Address bit
21	Not used	
22	A10	Most significant address bit
23	Not used	
24	Not used	
25	Not used	
26	Not used	
27	Not used	
28	Not used	
29	Not used	
30	Not used	
31	Not used	
32	Not used	
33	B0	Least significant COP object code bit
34	B7	Most significant COP object code bit
35	B2	Object code bit
36	B5	Object code bit
37	B3	Object code bit
38	B4	Object code bit
39	B6	Object code bit
40	B1	Object code bit
41	TRIGGER OUT	BREAKPOINT/TRACE indicator
42	Not used	
43	RST*	Same as RESET*
44	PROM DISABLE*	Select PROM or Shared Memory mode
45	See Note 1	
46	See Note 1	
47	V <sub>CC</sub>	+5V <sub>DC</sub> power from development system
48	V <sub>CC</sub>	+5V <sub>DC</sub> power from development system
49	GND	Power and signal return
50	GND	Power and signal return

**Note 1:** Pins 45 and 46 are used as follows:

PDS	with target board 980306552 REV A or later, normally not used. with target board 980305551 REV F or earlier, -12V <sub>DC</sub> from the PDS.
STARPLEX	with target board 980306254, normally not used. However, jumper W5 on the target board may be installed to supply -12V <sub>DC</sub> to the emulator board.

# Functional Verification of E24 Board Using COP2404

Note: This chapter describes the dual processor configuration (ROMless Microcontroller = COP2404) of the E24. When using the COP404, refer to Chapter 4.

Due to the board's physical location, external to the development system, it is easily damaged. Always observe the following:

1. User power supplies are adequately bypassed and supplying the correct voltage.
2. Development system power is not connected to user power.
3. Cables are correctly installed.
4. Device input/output ratings are not exceeded.
5. PROMs are in their correct sockets and properly oriented.
6. The COP4XX is receiving a valid clock signal.

If a mishap or malfunction does occur, National Semiconductor's Microcomputer Technical Support Manager can be contacted at (408) 721-6803. Questions concerning actual operation of the board or customer use of a COP400 device should be referred to the COPSTM Application Group at (408) 721-5582.

Alternatively, if the board develops a problem and circumstances do not allow sufficient time to send it back to National, there is a series of COPMON commands that may be used to isolate faulty components. Before attempting the following diagnostic aids, the user should study Section 5.5 of this manual and the schematic supplied with the emulator board. The user will also need a functional development system and an emulator board cable.

PDS users should study Chapter 9 of the PDS User's manual.

STARPLEX™ users should consult the SPM-A15 Operator's Manual (Manual No. 420306254) for information on COPMON.

1. With power turned off, connect the board to the development system, making sure all jumpers are correctly assigned. Correct jumper assignments are determined by the ROMless microcontroller on the board. Refer to Section 5.2.
2. Turn power on and load COPMON. When prompted, specify chip number as 2404.

Note: Underline indicates user inputs.

PDS Example:

```
CR
EXEC, REV:D
X> @COPMON
CHIP NUMBER (DEFAULT = 420)? 2440
C>
```

STARPLEX Example:

```
> COPMON
COPMON, REV: B, <date>
CHIP NUMBER (DEFAULT = 420)? 2440
C>
3. Load shared memory with CLRA (object code = X'00)
instructions.
C> DE 0, 0/L
4. Specify and perform a TRACE IMMEDIATE
C> TR I
TRACE ENABLED:
IMED OCCUR: 1 PRIOR: 0 GO: N
C> G
```

COPMON should come back with the following message:

```
TRACED ON IMED AT A:000
```

If it does not, then the CLK signal described in Section 5.5 is not being generated by the COP2404 and/or the CLK signal is not reaching the system. Probable faulty circuits:

```
COP4XX (U2)
74LS14 (U1)
```

Also, verify that the COP4XX is receiving a valid clock input.

If COPMON executed the TRACE properly, the COP program counter can now be examined with the TYPE command.

C> TY

0	0 A:000	E:1111	A:000	E:1111
2	2 A:000	E:1111	A:000	E:1111
4	4 A:000	E:1111	A:000	E:1111
6	6 A:000	E:1111	A:000	E:1111
8	8 A:000	E:1111	A:000	E:1111
10	10 A:000	E:1111	A:000	E:1111
12	12 A:000	E:1111	A:000	E:1111
14	14 A:000	E:1111	A:000	E:1111

Note all the address values (A:XXX) shown are zero. This is correct because the TRACE operation was begun before COPMON let RST\*/RESET\* go to a high level. If one or more of the program counter bits are stuck high, the TYPE command might yield the following information:

C> TY

0	0 A:009	E:1111	A:009	E:1111
2	2 A:009	E:1111	A:009	E:1111
4	4 A:009	E:1111	A:009	E:1111
6	6 A:009	E:1111	A:009	E:1111
8	8 A:009	E:1111	A:009	E:1111
10	10 A:009	E:1111	A:009	E:1111
12	12 A:009	E:1111	A:009	E:1111
14	14 A:009	E:1111	A:009	E:1111



With this information, the user can generally isolate which address line (A0-A10) is malfunctioning.

Probable faulty circuits:

- 81LS95 (U10)
- 81LS95 (U11)
- 74LS374 (U7)
- 74LS374 (U9)

5. Enter another TRACE IMMEDIATE command to test RST\*/RESET\* and proper binary operation of the address lines.

C>G

TRACED ON IMED AT A:28B

C>I

0	0		A:28B	E:1111
1	2 A:68B	E:1111	A:28C	E:1111
3	4 A:68C	E:1111	A:28D	E:1111
5	6 A:68D	E:1111	A:28E	E:1111
7	8 A:68E	E:1111	A:28F	E:1111
9	10 A:68F	E:1111	A:290	E:1111
11	12 A:690	E:1111	A:291	E:1111
13	14 A:691	E:1111	A:292	E:1111
15	15 A:692	E:1111		

C>TY

15	15		A:293	E:1111
17	17 A:693	E:1111	A:294	E:1111
19	19 A:394	E:1111	A:295	E:1111
21	21 A:695	E:1111	A:296	E:1111
23	23 A:696	E:1111	A:297	E:1111
25	25 A:697	E:1111	A:298	E:1111
27	27 A:698	E:1111	A:299	E:1111
29	29 A:699	E:1111	A:29A	E:1111
31	31 A:69A	E:1111		

If the RST\*/RESET\* line is stuck low, the addresses shown above would have remained at zero. Probable faulty circuits:

COP2404 (U2)

Development System

The COP addresses from this second TRACE IMMEDIATE operation should be inspected for monotonically increasing binary values from 0 to 7FFH and wraparound from this address to 0. This can be done by additional TRACE and TYPE commands.

If several address lines are shorted or inoperative, a TYPE command might yield program counter values like the following:

C>I

0	0 A:6C9	E:1111	A:2CB	E:1111
2	2 A:6CB	E:1111	A:2CB	E:1111
4	4 A:6CB	E:1111	A:2CD	E:1111
6	6 A:6CD	E:1111	A:2CD	E:1111
8	8 A:6CD	E:1111	A:2CF	E:1111
10	10 A:6CF	E:1111	A:2CF	E:1111
12	12 A:6CF	E:1111	A:2C9	E:1111
14	14 A:6C9	E:1111	A:2C9	E:1111

C>TY

16	16 A:6C9	E:1111	A:2CB	E:1111
18	18 A:6CB	E:1111	A:2CB	E:1111
20	20 A:6CB	E:1111	A:2CD	E:1111
22	22 A:6CD	E:1111	A:2CD	E:1111
24	24 A:6CD	E:1111	A:2CF	E:1111
26	26 A:6CF	E:1111	A:2CF	E:1111
28	28 A:6CF	E:1111	A:2D9	E:1111
30	30 A:6D9	E:1111	A:2D9	E:1111

Probable faulty circuits:

- 81LS95 (U10)
- 81LS95 (U11)
- 74LS374 (U7)
- 74LS374 (U9)

6. Given proper operation of the board to this point, test the actual program data bits (B0-B7) by inserting various jump commands into memory and using TRACE to verify that the jump occurred. If the board fails any of the following tests, the probable faulty circuit is:

81LS95 (U6)

Insert a jump to location 0 at address 3E; set up, execute, and list the trace.

C>PU 3E, JP 0

C>TR 3E

TRACED ON A:03E AT A:03E

A:03E OCCUR: 1 PRIOR: 0 GO: N

C>G

TRACED ON A:03E AT A:03E

C>I

0	0 A:03E	E:1111	A:43F	E:1111
2	2 A:000	E:1111	A:440	E:1111
4	4 A:001	E:1111	A:441	E:1111
6	6 A:002	E:1111	A:442	E:1111
8	8 A:003	E:1111	A:443	E:1111
10	10 A:004	E:1111	A:444	E:1111
12	12 A:005	E:1111	A:445	E:1111
14	14 A:006	E:1111	A:446	E:1111

If TRACE memory location 2 does not contain A:000, then the board is not recognizing the JP 0 instruction (object code = C0H) properly. If JP 0 is working, then high levels on B6 and B7 from the development system are being recognized by the board. Proper high levels on B0-B5 may now be tested by successively replacing the JP 0 instruction with the following jumps: JP 1 (C1H), JP 2 (C2H), JP 4 (C4H), JP 8 (C8H), JP 10 (D0H), and JP 20 (E0H). The necessary COPMON commands are:

C>PU 3E, JP 1

C>G

TRACED ON A:03E AT A:03E

```
C>T
0 0A:03E E:1111 A:43F E:1111
2 2A:001 E:1111 A:440 E:1111
4 4A:002 E:1111 A:441 E:1111
6 6A:003 E:1111 A:442 E:1111
8 8A:004 E:1111 A:443 E:1111
10 10A:005 E:1111 A:444 E:1111
12 12A:006 E:1111 A:445 E:1111
14 14A:007 E:1111 A:446 E:1111
```

Note that the contents of trace location 2 should be A:001. If B0 and B1 were shorted or inoperative, the TYPE command might yield the following information:

```
C>G
TRACED ON A:03E AT A:03E
```

```
C>T
0 0A:03E E:1111 A:43F E:1111
2 2A:003 E:1111 A:440 E:1111
4 4A:004 E:1111 A:441 E:1111
6 6A:005 E:1111 A:442 E:1111
8 8A:006 E:1111 A:443 E:1111
10 10A:007 E:1111 A:444 E:1111
12 12A:008 E:1111 A:445 E:1111
14 14A:009 E:1111 A:446 E:1111
```

Continuing the test:

```
C>TR 3E, 1, 0
```

TRACE ENABLED:

```
A:03E OCCUR: 1 PRIOR: 0 GO: N
```

```
C>PU 3E, JP 2
```

```
C>G
```

```
TRACED ON A:03E AT A:03E
```

```
C>T 0/9
```

```
0 0A:03E E:1111 A:43F E:1111
2 2A:002 E:1111 A:440 E:1111
4 4A:003 E:1111 A:441 E:1111
6 6A:004 E:1111 A:442 E:1111
8 8A:005 E:1111 A:443 E:1111
```

```
C>PU 3E, JP 4
```

```
C>G
```

```
TRACED ON A:03E AT A:03E
```

```
C>T 0/9
```

```
0 0A:03E E:1111 A:43F E:1111
2 2A:004 E:1111 A:440 E:1111
4 4A:005 E:1111 A:441 E:1111
6 6A:006 E:1111 A:442 E:1111
8 8A:007 E:1111 A:443 E:1111
```

```
C>PU 3E, JP 8
```

```
C>G
```

```
TRACED ON A:03E AT A:03E
```

```
C>T 0/9
```

```
0 0A:03E E:1111 A:43F E:1111
2 2A:008 E:1111 A:440 E:1111
4 4A:009 E:1111 A:441 E:1111
6 6A:00A E:1111 A:442 E:1111
8 8A:00B E:1111 A:443 E:1111
```

```
C>PU 3E, JP 10
```

```
C>G
```

```
TRACED ON A:03E AT A:03E
```

```
C>T 0/9
```

```
0 0A:03E E:1111 A:43F E:1111
2 2A:010 E:1111 A:440 E:1111
4 4A:011 E:1111 A:441 E:1111
6 6A:012 E:1111 A:442 E:1111
8 8A:013 E:1111 A:443 E:1111
```

```
C>PU 3E, JP 20
```

```
C>G
```

```
TRACED ON A:03E AT A:03E
```

```
C>T 0/9
```

```
0 0A:03E E:1111 A:43F E:1111
2 2A:020 E:1111 A:440 E:1111
4 4A:021 E:1111 A:441 E:1111
6 6A:022 E:1111 A:442 E:1111
8 8A:023 E:1111 A:443 E:1111
```

The final test ensures that B6 and B7 are not shorted together and that the SKIP line is functioning. Use a 2-byte JMP 1 (object code = 6001H) instruction.

```
C>PU 3E, JMP 1
```

```
C>G
```

```
TRACED ON A:03E AT A:03E
```

```
C>T 0/9
```

```
0 0A:03E E:1111 A:43F E:1111
2 2A:03F SKIP E:1111 A:440 E:1111
4 4A:001 E:1111 A:441 E:1111
6 6A:002 E:1111 A:442 E:1111
8 8A:003 E:1111 A:443 E:1111
```

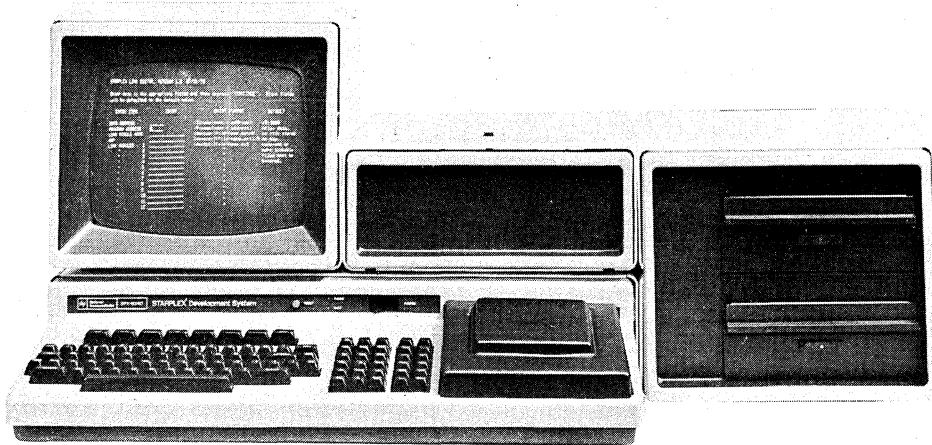
If trace memory location 4 does not contain A:001, then the JMP 1 instruction was not recognized. Also, if the jump to 001 was made, but location 2 does not show a skip, there is a problem with the SKIP line. Check:

```
74LS374 (U9)
```

```
81LS95 (U6)
```

If the emulator passes all the above tests, the supporting circuitry to the COP4XX is functional. It is imperative that all these tests be performed with a completely operational development system. If the user still experiences difficulties during program emulation, the Microcomputer Technical Support Manager, (408) 721-6803, should be contacted.

# STARPLEX™ Development System



## ■ A Total Development System

- Hardware:  
CPU, Floppy Discs, Video Monitor,  
Keyboard
- Software:  
Disc Operating System, Debugger,  
Editor, Macro Assembler, FORTRAN,  
BASIC, On-Board ROM Diagnostic  
and Utilities
- Options:  
Emulators, PROM programmer,  
Printers, STARLINK™, Cross  
Assemblers

## ■ Easy to Use

- Function keys direct system
- Prompting menus guide operator  
entries
- Comprehensible error messages
- Keystroke-driven editor

## ■ Full Product Line Support

- Supports 8080, 8048, 8049, 8050,  
NSC800, 8085, 8070 and Z-80-based  
microprocessor systems
- Expandable with industry standard  
BLC/SBC boards

## Product Overview

The STARPLEX™ Development System is a general purpose microcomputer and microprocessor development system. New levels of operating simplicity have been designed into the STARPLEX system to significantly reduce the amount of time spent on product development. By getting the user into actual application work sooner and with fewer mistakes, the STARPLEX system allows the user to take full advantage of time spent at the console.

### A Total System

The STARPLEX design combines all the components required for the entire development

task in one complete system. The STARPLEX package includes an 8080-based CPU board, 64K bytes of RAM, 1M byte of disc storage, a video monitor and keyboard. The standard STARPLEX software package includes a disc operating system, assembler, debugger, editor, linker, loader, FORTRAN, BASIC, on-board ROM diagnostic and utilities. Options available are: an in-system emulator for real-time debugging of customized hardware and software, PROM programmer personality modules for programming, verifying and copying PROMs, STARLINK™ for transferring files between STARPLEX™ and MDS Development System, and cross assemblers.

## Easy to Use

The STARPLEX System reduces the time a user must spend at a terminal by making many complex functions accessible through one easy keystroke. System commands are initiated by clearly marked function keys which invoke prompting menus to guide the user through each task. These function keys eliminate the need to memorize system commands and various command options. As a result, there is no need to refer to lengthy documentation, and errors or delays caused by incorrectly entered commands are eliminated.

Recognizing that a great deal of the user's time is spent on creating and changing source code, the designers of the STARPLEX system have devoted special attention to the text editing facility.

A set of special function keys directs the STARPLEX editor, allowing corrections to be made with single keystrokes. Also, the powerful "string mode" commands allow search and replacement of character strings as well as block moves. An entire file may be quickly and easily reviewed or altered. The number of mistakes is reduced because the data and changes are immediately displayed. Backup files are automatically created, protecting the user from accidental loss of data. Because the STARPLEX system is easy to use, learning time is considerably shortened. A first time user can be productive within a half hour. Also, as users make more efficient use of the system, machine availability is maximized.

## Full Product Line Support

When a user buys a STARPLEX System he can be assured it will meet both today's and tomorrow's development needs. All the boards within the STARPLEX System are members of National's Series/80 family and use the standard Series/80 bus, making the system expandable with the more than 40 boards presently available in this series.

The STARPLEX System supports development for the 8080A, 8048, 8049, 8050, NSC800, 8070, Z80 and 8085 processors and will support all future National and other popular microcomputer and microprocessor products.

## The Result — Cost Effectiveness

The most important feature of the STARPLEX System is that it saves development time. Its ease of use allows the designer to concentrate on solving the application problem, rather than learning how to operate the system. With the STARPLEX system, the effectiveness of a company's most valuable resource — "engineering manpower" — is maximized.

## Functional Description

### Hardware Modules

STARPLEX components are packaged into modules which form a unified system when placed together. The modules are durable, with housings constructed of 1/8 inch aluminum and front panels of molded lexan foam.

STARPLEX is designed for easy maintenance. Snap-down doors on the base module make it easy to access the card cages and circuit boards. Interconnecting cables between all modules and boards are routed to the rear of the system and covered by easily removable cable channels. Thus, cables are out of sight and protected from accidental damage. All cables, including the single AC power distribution system, are plug detachable at both ends, making it easy to disconnect modules and reconfigure the system.

Human engineering concepts have directed the design of each STARPLEX module to make the man-machine interface as natural as possible. For example, the video monitor screen has antireflective coating to minimize glare, and light-emitting diodes in certain keys provide operator awareness of their selection. Even cooling fans have been located to minimize noise levels.

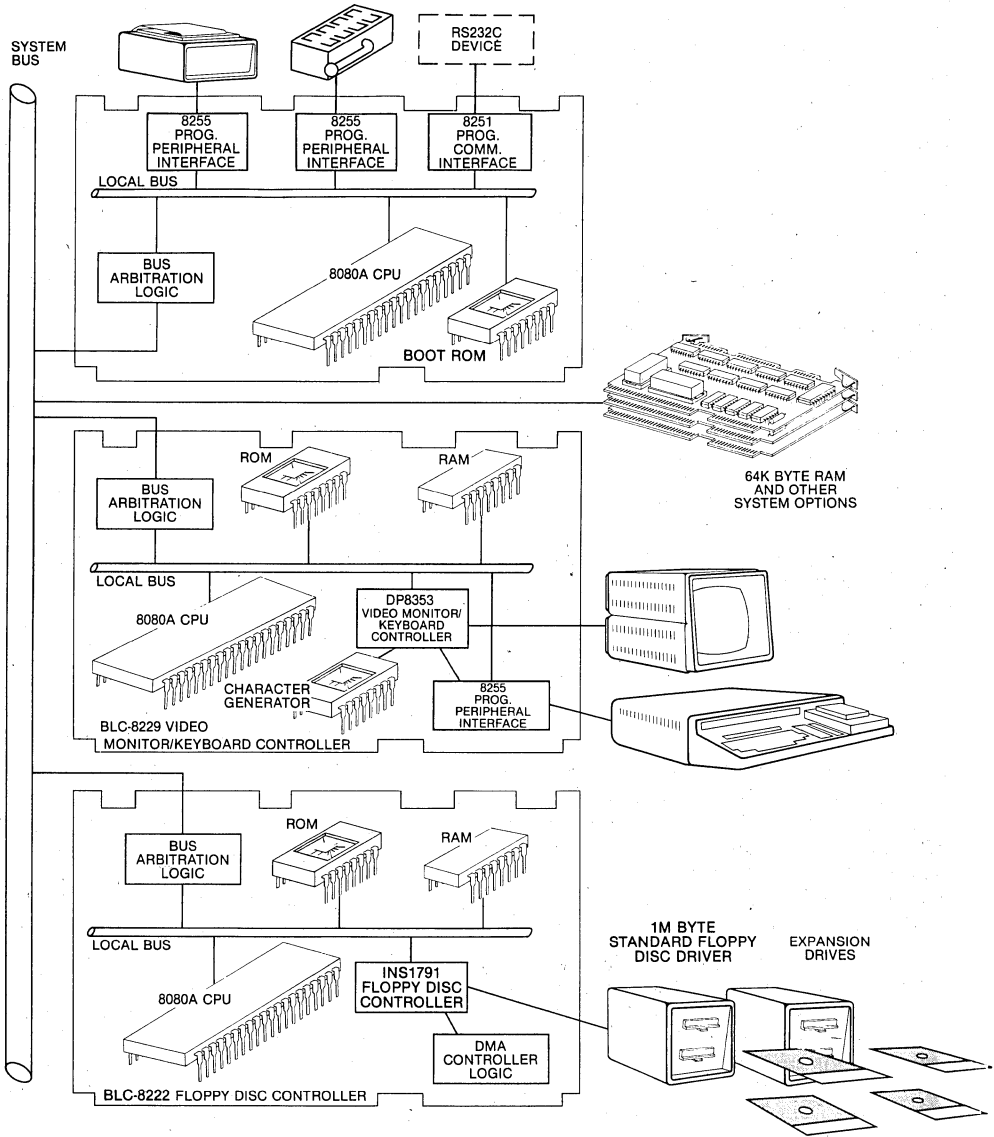
### STARPLEX Electronics

Four Series/80 boards make up the STARPLEX electronics: the main CPU board (based on the BLC-80/204), the video monitor/keyboard controller (BLC-8229), the floppy disc controller (BLC-8222) and a 64K memory board (BLC-064).

These boards communicate with each other via the standard BLC system bus. The CPU, BLC-8229 and BLC-8222 all have multi-master bus logic on their respective boards allowing them to share the system bus. The BLC-8229 and BLC-8222 communicate with the CPU using Direct Memory Access and programmed I/O.

The optional printers and PROM programmer personality modules communicate with the CPU through two programmable parallel I/O ports. An RS232C port on the CPU is available and permits both asynchronous and synchronous communications for use with a printer or a communications link such as STARLINK.

Individual circuit boards are built to National's high manufacturing quality standards, utilizing techniques such as computer aided layout and auto insertion. All boards and the system as a whole are tested dynamically under system load conditions at elevated temperatures as part of a thorough factory burn-in.



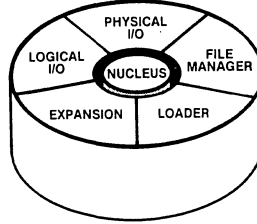
STARPLEX Multiprocessor System Diagram

**Software**

STARPLEX software is completely thought out from a functional standpoint, carefully engineered to be easy to understand and use and thoroughly integrated into the total system. Every aspect is designed to assist the user in rapidly developing microprocessor-based systems from the ground up.

The elegance of STARPLEX software lies in its ability to make the complicated process of program development appear simple to the user.

The software system is structured as a series of rings around a nucleus. Segments of these rings can be changed or added for future development requirements such as other high-level languages, file handlers or special user-defined routines.



**LEVEL I**

Level I of the operating system provides system housekeeping functions and coordinates access to system resources. It includes a file manager, an I/O control system and a loader.

*File Manager*

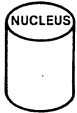
The file manager organizes, stores and retrieves data and programs stored on the diskette.

- Maintains a directory
- Allows multiple file attributes
- Uses a "hierarchical linked list" structure
- Supports random access

*I/O Control System*

The I/O control system is designed to eliminate the need for the user to understand the physical I/O characteristics of each individual device and presents a simplified, logical device-independent architecture.

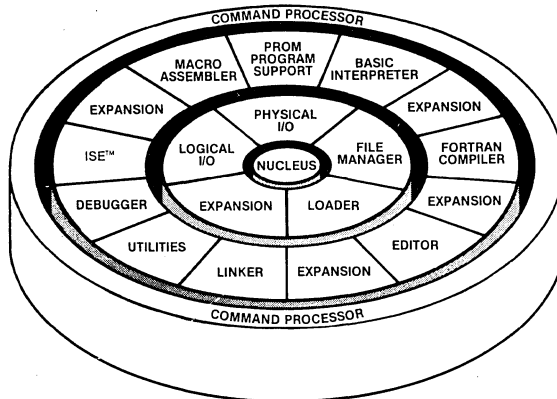
- Provides overlapped I/O commands
- Allows files to be accessed by name
- Handles error conditions



**NUCLEUS**

The nucleus of the STARPLEX operating system controls and allocates system resources for the higher level processes.

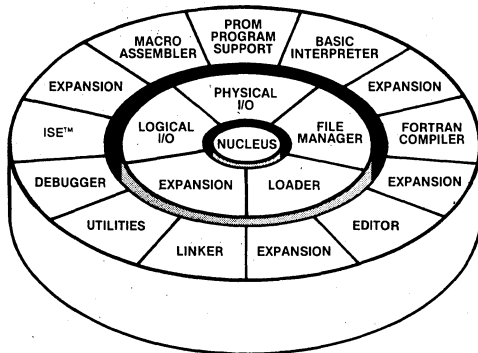
- Provides synchronization and communication facilities for higher level asynchronous processes
- Services all hardware interrupts
- Provides interval timer functions
- Completely device-independent



### Loader

The loader brings programs into main memory at specified locations.

- Provides "load and go" mode
- Allows controlled load mode — starting address returned to calling program



### LEVEL 2

Level 2 of the operating system provides the "development services" including a linker, a CRT-oriented editor, utilities, a debugger, PROM programmer support, Macro assemblers, BASIC and FORTRAN IV.

### Linker

The linker combines selected relocatable object modules created by the assembler or language compiler into an executable run time module.

- Assigns absolute addresses to load modules
- Produces a memory map of linked components
- Searches system and user libraries for unresolved external references

### Editor

The STARPLEX editor is an easy-to-use CRT-oriented text editor.

- Function key driven
- String search and replace
- Forward and backward paging
- Block moves
- Automatic source file backup
- Traps illegal commands

### Utilities

General utilities provide routine maintenance functions.

- Transfer data files between devices
- Obtain diskette directory listings
- Format diskettes
- Modify file attributes
- Rename files

### Debugger

The program debugger simplifies 8080 program checkout by allowing program execution to be monitored and altered.

- Allows single step control
- Permits eight breakpoint assignments
- Displays program counter and registers at breakpoints
- Memory references are absolute or relative to one of the relocation registers

### PROM Programmer Support

The PROM programmer support software manages the optional PROM personality module functions.

- Allows PROM code to be listed, verified and copied
- Data stored in a PROM can be transferred to or from another PROM, a diskette file, memory, the video monitor or keyboard.

### Macro Assemblers

The Macro assemblers assemble 8080, 8085, 8048, 8070, NSC800, and Z80 mnemonic code and allow operator definition of useful higher level instructions called "Macros" which are then expanded into a sequence of machine level instructions.

- Generates absolute or relocatable object modules
- Conditional assembly parameters
- Allows external references

### FORTRAN IV

The FORTRAN IV compiler on the STARPLEX system meets the ANSI X3.9-1966 standard and includes the following enhancements:

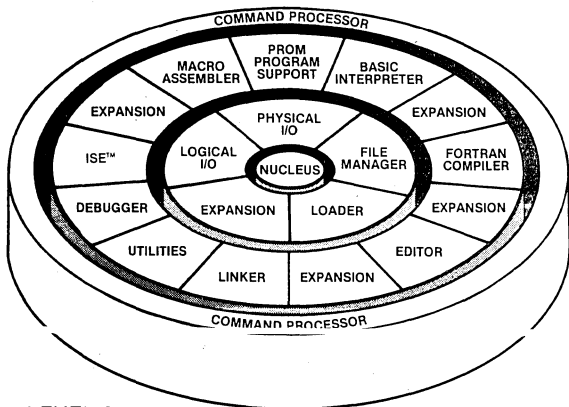
- PEEK and POKE — allow direct access to memory
- INP and OUT — allow direct I/O access

- Comprehensive subroutine library
- Supports user-written I/O drivers
- Random access disc I/O
- Allows assembly language subroutine calls

**BASIC**

The STARPLEX BASIC compiler/interpreter conforms to the Dartmouth defined BASIC with extensions:

- PEEK and POKE — allow direct access to memory
- INP and OUT — allow direct I/O access for non-STARPLEX devices
- Complete string operators
- Multi-dimensional arrays
- Extensive debugging and programming aids — trace, edit, direct mode, renumber



**LEVEL 3**

The Command Interpreter is the interface between the operating system and the human operator.

- Function key driven
- Verifies user requests
- Provides menus and prompting for system commands

**Specifications**

- Memory — 64K bytes
- Floppy Disc —
  - Format IBM compatible, soft-sectored
  - Capacity 512K bytes per drive
  - Maximum 1 million bytes
  - Capacity (4 drives)

**Printers —**

Type	Thermal
Speed	50 characters per second
Width	80 columns
Character Type	5x7 dot matrix
Type	Impact
Speed	120 characters per second
Width	132 columns
Character Type	7 x 9 dot matrix

**Video Monitor —**

Matrix	7x9 dot
Display Array	80 columns by 24 lines
Phosphor	P2 green

**Power —**

115VAC, 60Hz, 10 amps (max)  
or  
230VAC, 50Hz, 5 amps (max)

Base Module 644 Watts

Floppy Disc Module 966 Watts

Thermal Printer 126 Watts

Impact Printer 360 Watts

Video Monitor 34 Watts

**Physical —**

	Base Module	Floppy Disc Module	Thermal Printer	Impact Printer	Video Monitor
Height	5.75 in. 14.6 cm	11.5 in. 29.2 cm	5.75 in. 14.6 cm	8 in. 20.3 cm	11.5 in. 29.2 cm
Width	26 in. 66 cm	13 in. 33 cm	13 in. 33 cm	24.5 in. 62.2 cm	13 in. 33 cm
Depth	26 in. 66 cm	.19 in. 48.3 cm	19 in. 48.3 cm	18 in. 45.7 cm	19 in. 48.3 cm
Weight	68 lb. 30.8 kg	50 lb. 22.7 kg	28 lb. 12.7 kg	60 lb. 27 kg	29 lb. 13.2 kg



**Order Information**

SPX-80/40	STARPLEX Development System with Thermal Printer (60 Hz)
SPX-80/41	STARPLEX Development System without Printer (60 Hz)
SPX-80/42	STARPLEX Development System with Impact Printer (60 Hz)
SPX-80/51	STARPLEX Development System with 1 Megabyte Disc Storage (60 Hz)
SPX-80/61	STARPLEX Development System with 2 Megabyte Disc Storage (60 Hz)

**Note:** To order 50Hz add the letter "E" to the order number.

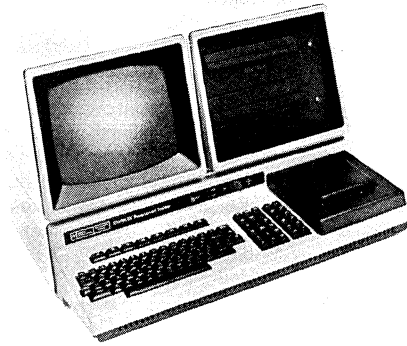
**Options**

SPM-A02	PROM programming interface plus software
SPM-A02-1	PROM programming module for programming 2708 EPROMs
SPM-A02-2	PROM programming module for programming 2716 EPROMs
SPM-A06-1	STARPLEX 1 Megabyte Dual Disc Expansion
SPM-A06-2	STARPLEX 2 Megabyte Dual Disc Expansion
SPM-A08	In-System Emulator Module
SPM-A09-1	8080 Emulator Package
SPM-A09-2	8048 Emulator Package
SPM-A09-3	8070 Emulator Package
SPM-A10	Z-80 Development Package
SPM-A15	COPS In-System Emulator (ISE) Package
SPM-A50	Thermal Printer
SPM-A55	Impact Printer
SFW-A001-1C	8048 Cross-Assembler
SFW-A002-1C	8070 Cross-Assembler
SFW-A003-1C	NSC800 Cross-Assembler
AEE-A001	STARLINK — SPX/MDS220, 230 Link
AEE-A002	STARLINK — SPX/MDS800, 888 Link

**Documentation**

420305546-001	STARPLEX System Reference Manual
420305788-001	STARPLEX System Software Reference Manual
420305789-001	STARPLEX Macro Assembler Software User's Manual
420305790-001	STARPLEX FORTRAN Compiler Software User's Manual
420305791-001	STARPLEX BASIC Interpreter Software User's Manual
420305586-001	BLC-8201/8221 Floppy Disc Controller Hardware Reference Manual
420305804-001	BLC-8222 Double Density Floppy Disc Controller Hardware Reference Manual
420305587-001	BLC-8228/8229 Video Monitor/Keyboard Controller Hardware Reference Manual
420305793-001	STARPLEX Hardware Maintenance Manual
420305521-001	BLC-80/204 Board Level Computer Hardware Reference Manual
420305529-001	BLC-032/048/064 32/48/64K RAM Board Hardware Reference Manual
420305869-001	In-System Emulator Reference Manual
420305653-001	SPM-A09-1 8080 ISE Target Board User's Manual
420306065-001	SPM-A09-2 8048 ISE Target Board User's Manual
420306155-001	SPM-A10 ZSTAR™ Z-80 Development System Reference Manual
420306154-001	Z80 Assembler Manual
420306064-001	8048 Family Cross-Assembler User's Manual
420306469-001	COP400 Emulator Card User's Manual
420306253-001	COP Cross-Assembler User's Manual
420306254-001	COP ISE Operator's Manual

# STARPLEX II™ Development System



## ■ A Complete Development System

- Dual CPU microprocessor-based system in master/slave configuration
  - 128K bytes of Random Access Memory
  - Dual floppy disk drives
  - Video monitor and keyboard controller
  - Two RS232C interfaces
  - Integral CRT keyboard with eight upper/lower case for a total of sixteen user definable keys
  - PROM programmer interface
- Software
  - Disk Operating System
  - Resident Debugger
  - Text Editor
  - Macro Assembler
  - On-board ROM and RAM diagnostics
  - I/O Spooling
  - FORTRAN
  - BASIC
- Options
  - In-System Emulator (ISE™) packages for NSC800, INS8048 family, INS8070 family, NS80CX48, 8080, 8085 and Z80® microprocessor devices
  - In-System Emulator package for COP400 microcontroller devices
  - PL/M for 8080/8085, PL/M for NSC800/Z80®
  - PASCAL compiler for 8080/8085; PASCAL compiler for NSC800/Z80®

- Optional double-sided/double-density disk drives with 2 megabytes of memory expandable to 4 megabytes
- Cross assemblers (Included with the emulator packages)
- Quiklook™ Tester to perform incoming inspection for COP400 microcontroller devices
- STARLINK™ — Interface to Intellec™ Development System
- PAL™/PROM programmer personality modules

## ■ Field-Upgradable from STARPLEX™ 80/41, 80/51 or 80/61 Systems

- Upgrade kit includes:
  - Z80A Master CPU Board
  - Z80A Slave CPU Board with 64K bytes of RAM
  - Internal RS232C cable and connector
  - Keyboard with user-definable keys
  - Disk-Based Operating System for STARPLEX II

## ■ Easy to Use

- Prompting menus guide operator entries
  - English language explanation of user errors
  - Direct system function keys to PAUSE/CONTINUE/ABORT/DEBUG
  - HELP key for online user assistance
  - Single stroke CRT edit keys

## Product Overview

The STARPLEX II Development System is a general-purpose microcomputer and microprocessor development system. New levels of operating simplicity have been designed into the STARPLEX II system to significantly reduce the amount of time spent on product development. By getting the user into actual application work sooner and with fewer mistakes, the STARPLEX II system allows the user to take full advantage of time spent at the console.

### A Complete System

The STARPLEX II design combines all the components required for the entire development task in one complete system. The STARPLEX II package includes a Z80A-based system controller board, a Z80A-based user processor/memory board with 64K bytes of RAM, 64K bytes of system RAM, 1M byte of disk storage controlled by a floppy disk controller, a video monitor and keyboard. The standard STARPLEX II software package includes a disk operating system, Z80 assembler, debugger, editor, linker, loader, FORTRAN, BASIC, on-board ROM diagnostics and utilities. Options available are: in-system emulation packages for real-time debugging of customized hardware and software prototype systems, PAL/PROM programmer personality modules for verifying, copying and programming PROMs or PALs, STARLINK for transferring files between STARPLEX II and Intellec™ Development System, and cross assemblers.

### Easy to Use

The STARPLEX Systems reduce the time a user must spend at a terminal by making many complex functions accessible through single easy keystrokes. System commands are initiated by clearly marked function keys which invoke prompting menus to guide the user through each task. These function keys eliminate the need to memorize system commands and various command options. As a result, there is no need to refer to lengthy documentation, and errors or delays caused by incorrectly entered commands are eliminated. With the user-definable keys on the STARPLEX II System keyboard, the amount of time a user must spend at a terminal is further reduced. Eight function keys are provided with upper and lower case capability for a total of sixteen different keys which are user-definable. These keys may be utilized both in command mode (system) and by an application program running on the system. Thus, while system commands are initiated by clearly marked function keys, which invoke prompting menus to guide the user through each task, many non-system complex functions become accessible through these user-definable keys.

Recognizing that a great deal of the user's time is spent on creating and changing source code, the designers of the STARPLEX II system have devoted special attention to the text editing facility.

A set of special function keys direct the STARPLEX II Editor, allowing corrections to be made with single keystrokes. Also, the powerful "string mode" commands allow search and replacement of character strings as well as block moves. An entire file may be quickly and easily

reviewed or altered. The number of mistakes is reduced because the data and changes are immediately displayed. Backup files are automatically created, protecting the user from accidental loss of data. Because the STARPLEX II system is easy to use, learning time is considerably shortened. A first-time user can be productive within a half hour. Also, as users make more efficient use of the system, machine availability is maximized.

### Spoiled Printer Capability

STARPLEX II supports spooled I/O to a user-selected print or another input or output device. Thus, printing long listings of files, compiler output and similar tasks may now be done at the same time as text editing, compiling, emulation, debugging, etc. The net result is a greater utilization of designer resources and subsequent reduction in program development time.

### Resident System Debugger

The system debug utility is resident and always available to the user. This program does not occupy any user space in memory and can be invoked by a single keystroke. Unlike many other debug utilities, the STARPLEX II debugger does not have to be specified prior to program execution and may be invoked at any time.

### Full Product Line Support

When a user buys a STARPLEX II system, he can be assured it will meet both today's and tomorrow's development needs. All the boards within the STARPLEX II system use the standard Series/80 bus, allowing the system to be expandable with the family of boards presently available in the Series/80 BLC line or with options currently available for use on STARPLEX II, such as ISE 8085 (SPM-90-A13-3) or ISE NSC800 (SPM-90-A13-4), each with Integral ISE (SPM-90-A13).

The STARPLEX II system supports development for the NSC800, NS16000, INS8070 family (8070, 8072, 8073 with Tiny Basic Interpreter), INS8048 family (8048, 8049, 8050), NS80CX48, Z80A, Z80B, 8085 processors and COP400 microcontroller devices, and will support future National and other popular microcomputer and microprocessor based products.

## Functional Description

### Hardware Modules

STARPLEX II components are packaged into modules which form a unified system when placed together. The modules are durable, with housings constructed of 1/8-inch aluminum and front panels of molded lexan foam.

STARPLEX II is designed for easy maintenance. Snap-down doors on the base module make it easy to access the card cages and circuit boards. Interconnecting cables between all modules and boards are routed to the rear of the system and covered by easily removable cable channels. Thus, cables are out of sight and protected from accidental damage. All cables, including the single AC power distribution system, are plug-detachable at both ends, making it easy to disconnect modules and reconfigure the system as the user chooses.

## STARPLEX II Electronics

Five printed circuit boards make up the STARPLEX II electronics: the main Z80A-based CPU board, a Z80A-based user processor board which also has 64K bytes of memory, an 8080A-based video monitor/keyboard controller board, an 8080A-based floppy disk controller board and an additional 64K byte memory board.

The Z80A-based CPU board and user slave processor board are designed in a master/slave configuration to give the user processing power and speed that were unobtainable with previous development systems. The main CPU board with the floppy disk controller board and the video/keyboard controller board all have multi-master bus logic allowing them to share the system bus. The floppy disk controller board and the video/keyboard controller board communicate with the main CPU board and user processor board using Direct Memory Access and programmed I/O.

The optional printers and PAL™/PROM programmer personality modules communicate with the main CPU/user processor boards through two programmable parallel I/O ports. A pair of RS232C ports on the main CPU board are available and permit both asynchronous and synchronous communications for use with options such as STARLINK.

Individual circuit boards are built to National's high manufacturing quality standards, utilizing techniques such as computer-aided layout and auto insertion. All boards are tested dynamically under system load conditions at elevated temperatures as part of a thorough factory burn-in.

## Software

User programs are separated from those of the STARPLEX II operating system. This means that users have much more memory space available, and since the operating system resides in its own environment, accidental interface between user programs and the operating system is virtually eliminated.

The STARPLEX II software is completely thought out from a functional standpoint, carefully engineered to be easy to understand and use, and thoroughly integrated into the total system. Every aspect is designed to assist the user in rapidly developing microprocessor-based systems from the ground up.

The elegance of STARPLEX II software lies in its ability to make the complicated process of program development appear simple to the user.

## OPERATING SYSTEM

The operating system provides system housekeeping functions and coordinates access to system resources. It includes a nucleus file manager, an I/O control system and a loader.

The nucleus of the STARPLEX II operating system controls and allocates system resources for the higher-level processes. The nucleus:

- Provides synchronization and communication facilities for higher-level asynchronous processes.

- Services all hardware interrupts.
- Provides interval timer functions.
- Is completely device-independent.

## File Manager

The file manager organizes, stores and retrieves data and programs stored on the diskettes.

- Maintains a directory.
- Allows multiple file attributes.
- Supports random access.

## I/O Control System

The I/O control system is designed to eliminate the need for the user to understand the physical I/O characteristic of each individual device and presents a simplified, logical device-independent architecture.

- Provides overlapped I/O commands.
- Allows files to be accessed by name.
- Handles error conditions.
- Supports spooled I/O to a user-selected print or another input or output device

## Loader

The loader brings programs into main memory at specified locations.

- Provides "load and go" mode.
- Allows controlled load mode — starting address returned to calling program, useful for implementing overlay structures.

## DEVELOPMENT SERVICES

The "development services" include a linker, a CRT-oriented editor, utilities, a resident debugger, optional PAL/PROM programmer support macro assemblers, BASIC and FORTRAN IV, optional PL/M for NSC800/Z80 or 8080/8085, and optional PASCAL for NSC800/Z80 or 8080/8085.

## Linker

The linker combines relocatable object modules created by the assemblers or compilers into an executable run time module.

- Assigns absolute addresses to load modules.
- Produces a memory map of linked components.
- Searches system and user libraries for unresolved external references.

## Editor

The STARPLEX II editor is an easy-to-use CRT-oriented text editor.

- String search and replace.
- Forward and backward paging.
- Block moves.
- Automatic source file backup.
- Traps illegal commands.

## Utilities

General utilities provide routine maintenance functions.

- Transfer data files between devices.
- Obtain diskette directory listings.
- Format diskettes.

- Modify file attributes.
- Rename files.
- Print screen.

**Debugger**

The system debug utility is resident and always available to the user. The debugger does not occupy any user space in memory and may be invoked by a single key-stroke. The program debugger simplifies program check-out by allowing program execution to be monitored and altered.

- Allows single step control.
- Permits eight breakpoint assignments.
- Displays program counter and registers at breakpoints.
- Memory references are absolute or relative to one of the relocation registers.

**PAL/PROM Programmer Support**

The PAL/PROM programmer support software manages the optional PAL/PROM personality module functions.

- Allows PROM code to be listed, verified and copied.
- Data stored in a PROM can be transferred to or from another PROM, a diskette file, memory, the video monitor or keyboard.
- Allows for custom programming of programmable array logic devices (PAL).

**Macro Assembler**

Individual macro assemblers can assemble 8080, 8085, 8048, 8070, NSC800, or Z80 mnemonic code and allow operator definition of useful higher-level instructions called "Macros" which are then expanded into a sequence of machine-level instructions. (Macro assembler for NSC800/Z80 is included with the STARPLEX II system. All other cross assemblers are optional.)

- Generates absolute or relocatable object modules.
- Conditional assembly parameters.
- Allows external references.

**FORTAN IV**

The FORTRAN IV compiler on the STARPLEX II system meets the ANSI X3.9-1966 standard and includes the following enhancements:

- PEEK and POKE — allow direct access to memory.
- Supports user-written I/O drivers.
- Random access disk I/O.
- Allows assembly language subroutine calls.

**BASIC**

The STARPLEX II BASIC compiler/interpreter conforms to the Dartmouth-defined BASIC with extensions:

- PEEK and POKE — allow direct access to memory.
- Complete string operators.
- Multi-dimensional arrays.
- Extensive debugging and programming aids — trace, edit, direct mode, renumber.

**PL/M for 8080/8085 and NSC800/Z80 (Optional)**

PL/M is compatible with the industry standard PL/M, but offers many enhancements to improve program execution time and memory utilization.

- Available for 8080/8085 object code or NSC800/Z80 object code.

- Hardware access via high-level statements.
- Block structure facilitates structured programming techniques.
- Relocatable and linkable output object code.

*PASCAL for 8080/8085<sup>1</sup> and NSC800/Z80<sup>2</sup> (Optional)*

**Specifications**

**Processor Subsystem:** Z80A-based CPU board  
Z80A-based user processor/  
memory with 64K bytes RAM  
Video monitor/keyboard  
controller  
Double-density floppy disk  
controller  
Memory board with 64K bytes  
RAM (128K bytes total RAM)

**Floppy Disk Subsystem:**  
Configuration Dual disk drives  
Format IBM-compatible, soft-sector  
Capacity Double-density, single-sided  
512K bytes/drives  
Maximum Capacity Expanded to 4 double-density,  
double-sided drives with 4  
megabyte storage capacity

**Keyboard Subsystem:**  
System Function 8 single-stroke system  
control keys  
ASCII 58 alphanumeric keys  
Programmable 8 user-definable keys with  
upper/lower case

**CRT Subsystem:**  
Matrix 7×9 dot  
Display Array 80 columns by 25 lines  
Phosphor P2 green  
Other Screen tilted 10° for comfort-  
able viewing

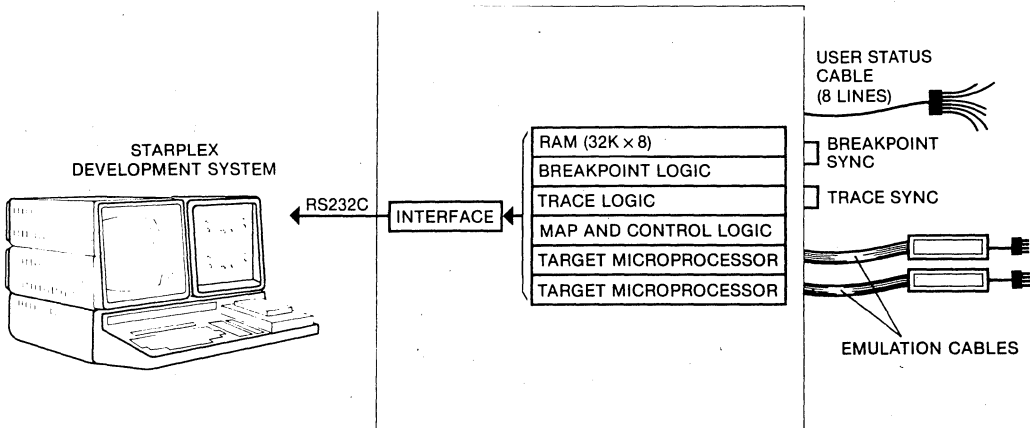
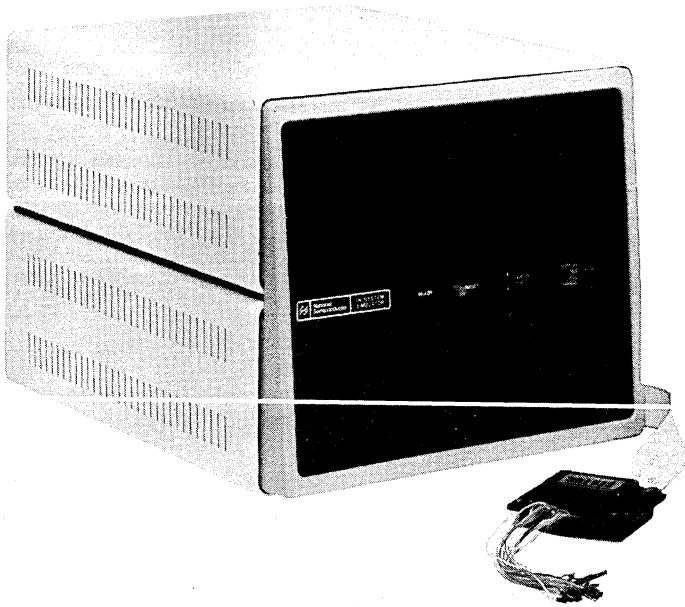
**Printers:**  
Type Impact  
Speed 120 characters per second  
Width 132 columns  
Character Type 7×9 dot matrix  
**Power:** 115 VAC, 60Hz, 10 amps (max)  
or  
230 VAC, 50 Hz, 5 amps (max)

Base Module 644 Watts  
Floppy Disk Module 966 Watts  
Impact Printers 360 Watts  
Video Monitor 34 Watts

**Physical:**

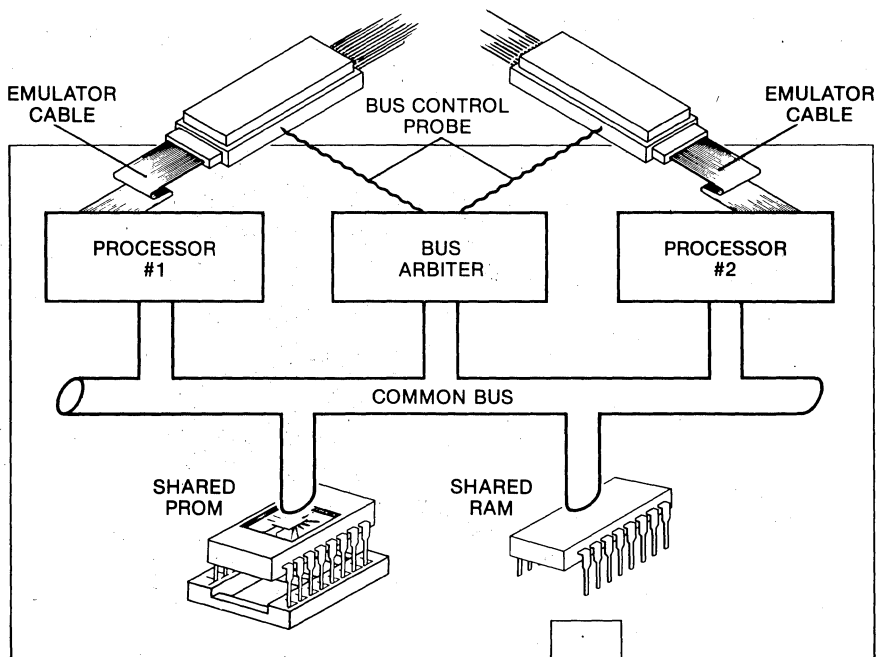
	Base Module	Floppy Disk Module	Impact Printer	Video Monitor
Height	5.75 in. 14.6 cm	11.5 in. 29.2 cm	8 in. 20.3 cm	11.5 in. 2.92 cm
Width	26 in. 66 cm	13 in. 33 cm	24.5 in. 62.2 cm	13 in. 33 cm
Depth	26 in. 66 cm	19 in. 48.3 cm	18 in. 45.7 cm	19 in. 48.3 cm
Weight	68 lb. 30.8 kg	50 lb. 22.7 kg	60 lb. 27 kg	29 lb. 13.2 kg

In-System Emulator Module

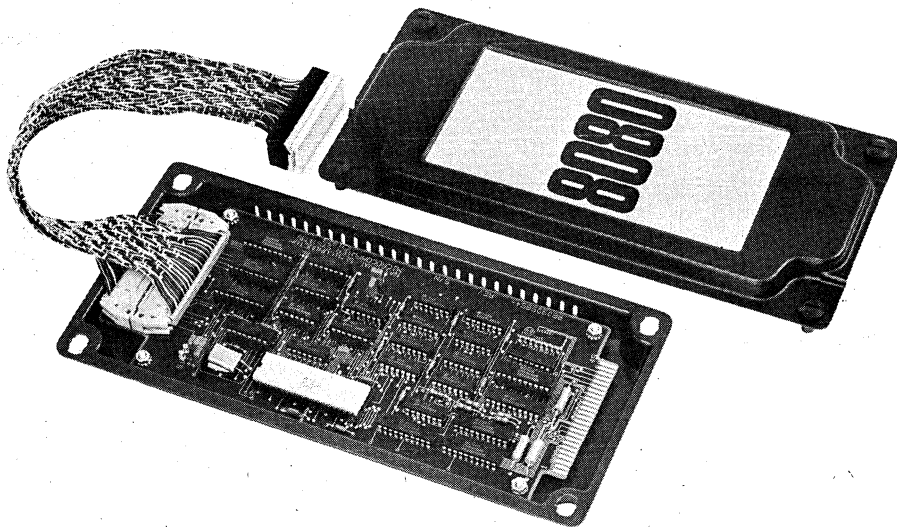


In-System Emulator System Configuration

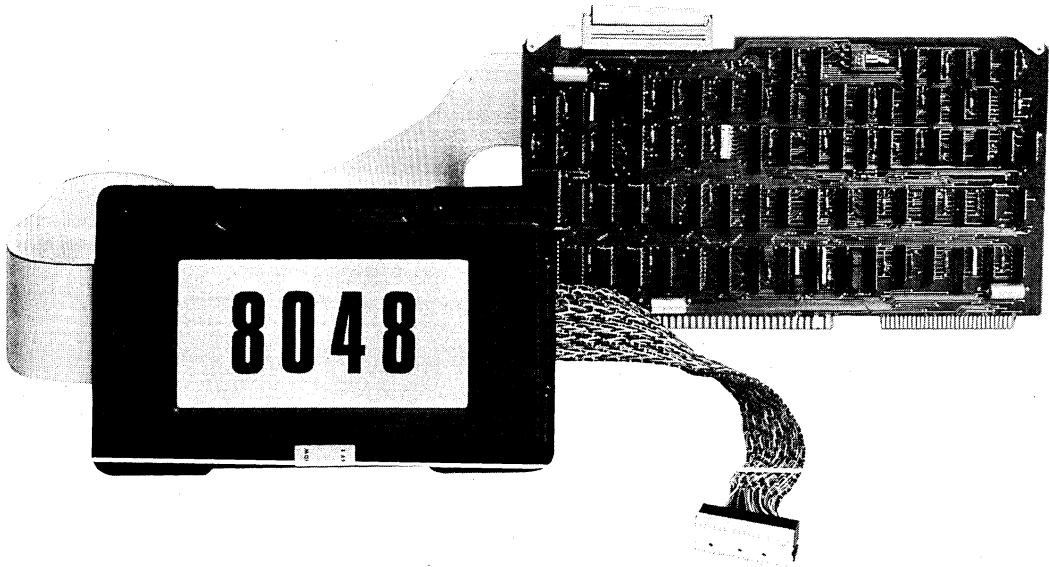
### Application Multiprocessor System Configuration



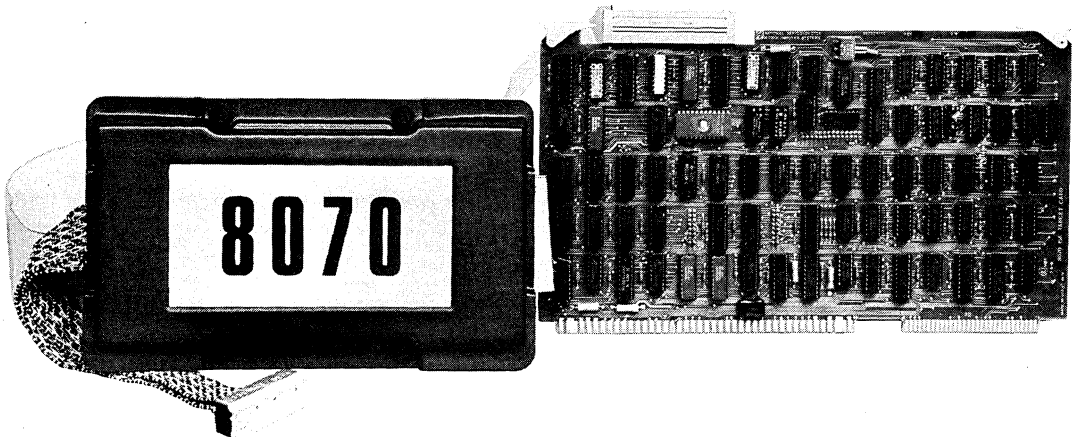
### 8080 Emulator Package



## 8048 Family Emulator Package

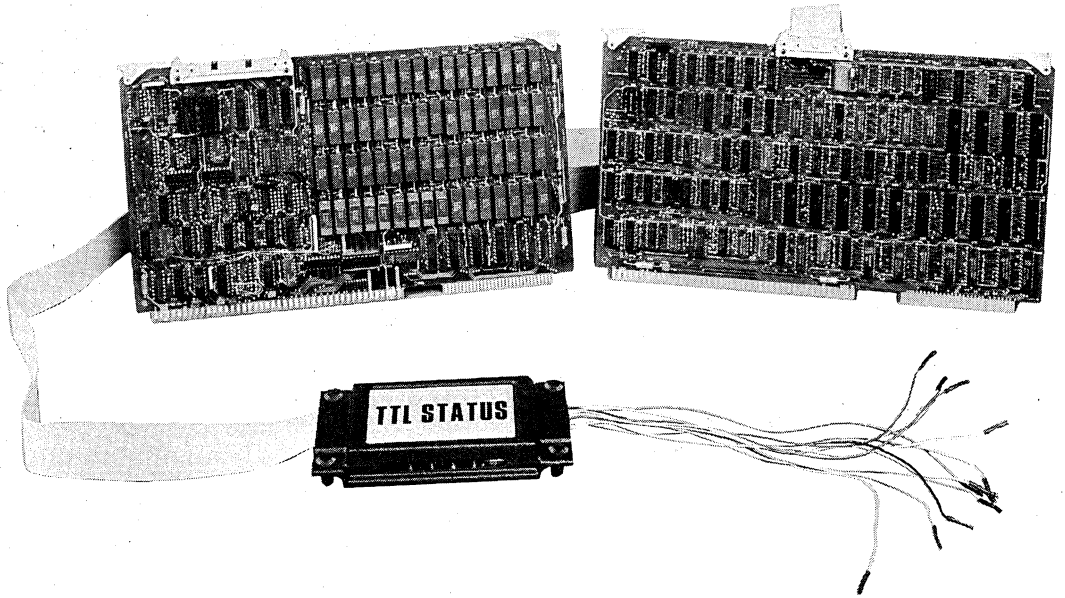


## 8070 Series Emulator Package

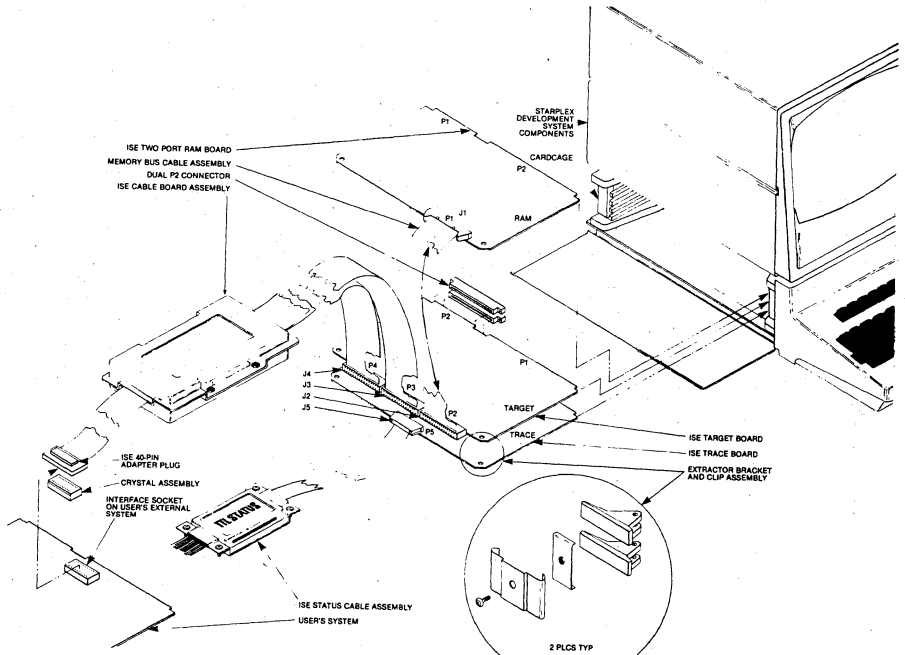




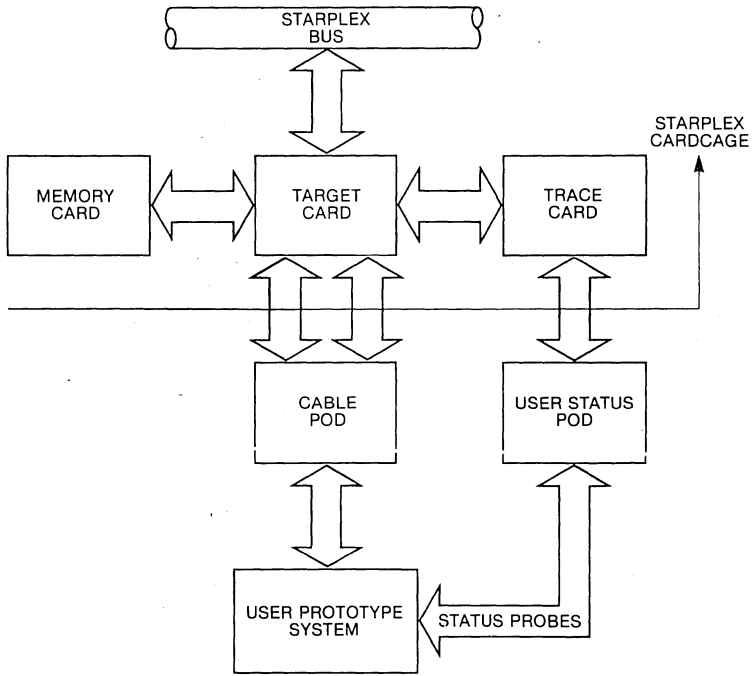
# Integral In-System Emulator



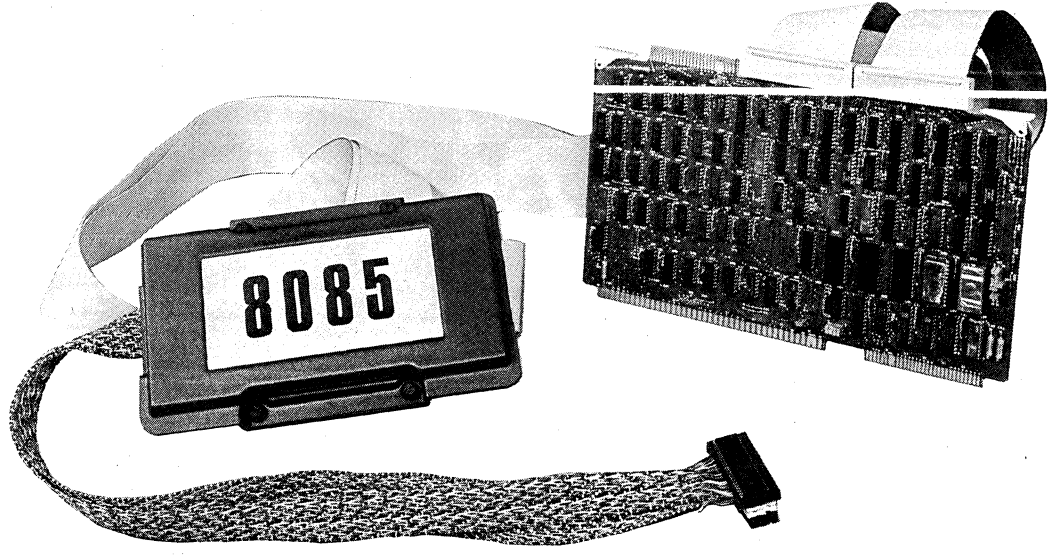
## Integral ISE Components Installation



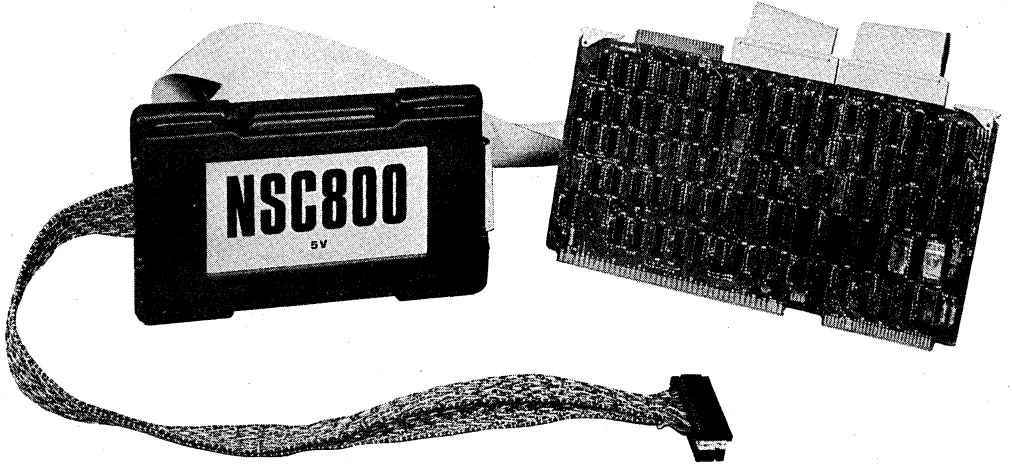
### Integral ISE System Configuration (Total: 3 Boards and 2 Pods)



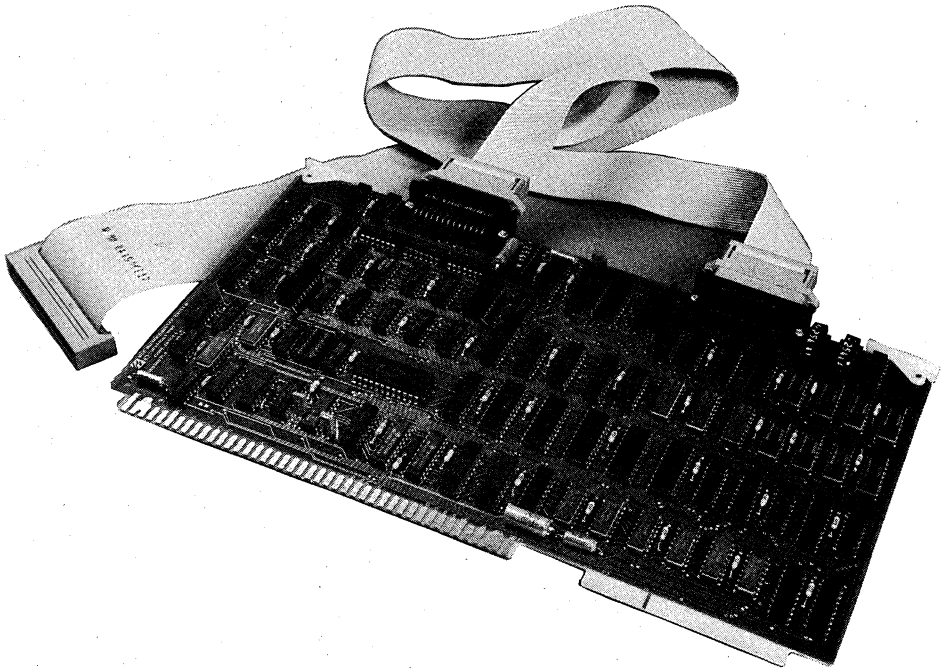
### 8085 Emulator Package



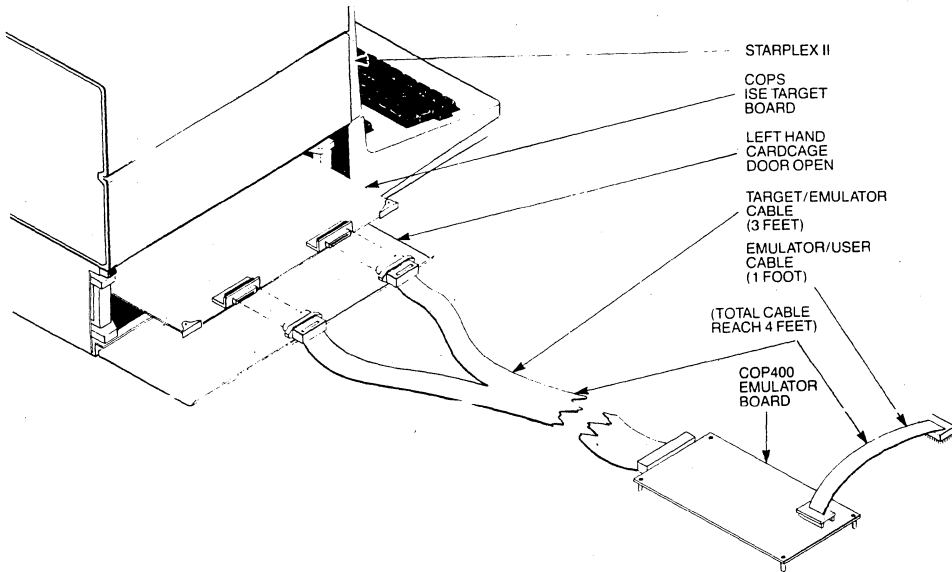
### NSC800 (5V) Emulator Package



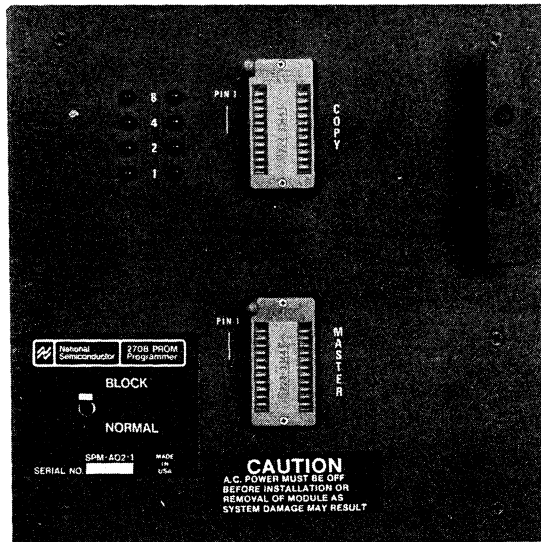
### COPS™ In-System Emulator Package



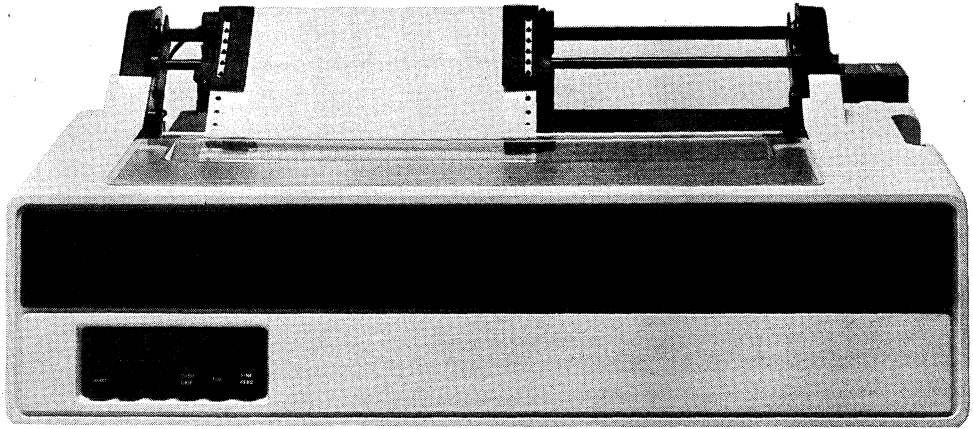
### Installation of the COPS ISE Target Board and an Emulator Board



### Universal PROM/PAL Programmer Personality Modules for 2708, 2716 EPROMs and PALs



# Impact Printer



## STARPLEX II Development System

### Video Monitor Subsystem

Large screen — measures 12" diagonally  
 Legible characters — 7×9 dot matrix  
 24 lines × 80 characters  
 Soft green phosphor  
 Variable screen intensity  
 10° tilted screen for comfortable viewing  
 Extensive screen control; scrolling,  
 blink, blank, inverse video or  
 alternate characters

### User Definable Function Keys

Eight function keys are provided  
 with upper and lower case  
 capability for a total of sixteen  
 different keys which are user  
 definable

### Processors Subsystem

Z80-based CPU  
 Z80-based user processor/memory  
 with 64K byte RAM  
 Floppy disk controller/formatter  
 64K byte RAM  
 Dual 4-slot chassis provides  
 three expansion slots

### Disk Subsystem

Dual standard floppy drives give  
 512K bytes per drive capacity  
 Uses IBM soft-sectored format  
 Expandable to four drives  
 (two million bytes)

### PROM Programmer (Optional)

Plug in PROM personality modules —  
 standard PRO-LOG compatible  
 Programs bipolar PROMs,  
 2708, 2716 EPROMs, PALs

### System Function Keypad

9 system control keys  
 Control program execution

### Editor Keypad

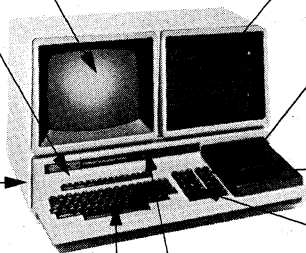
5 cursor keys  
 13 special edit keys

### ASCII Keypad

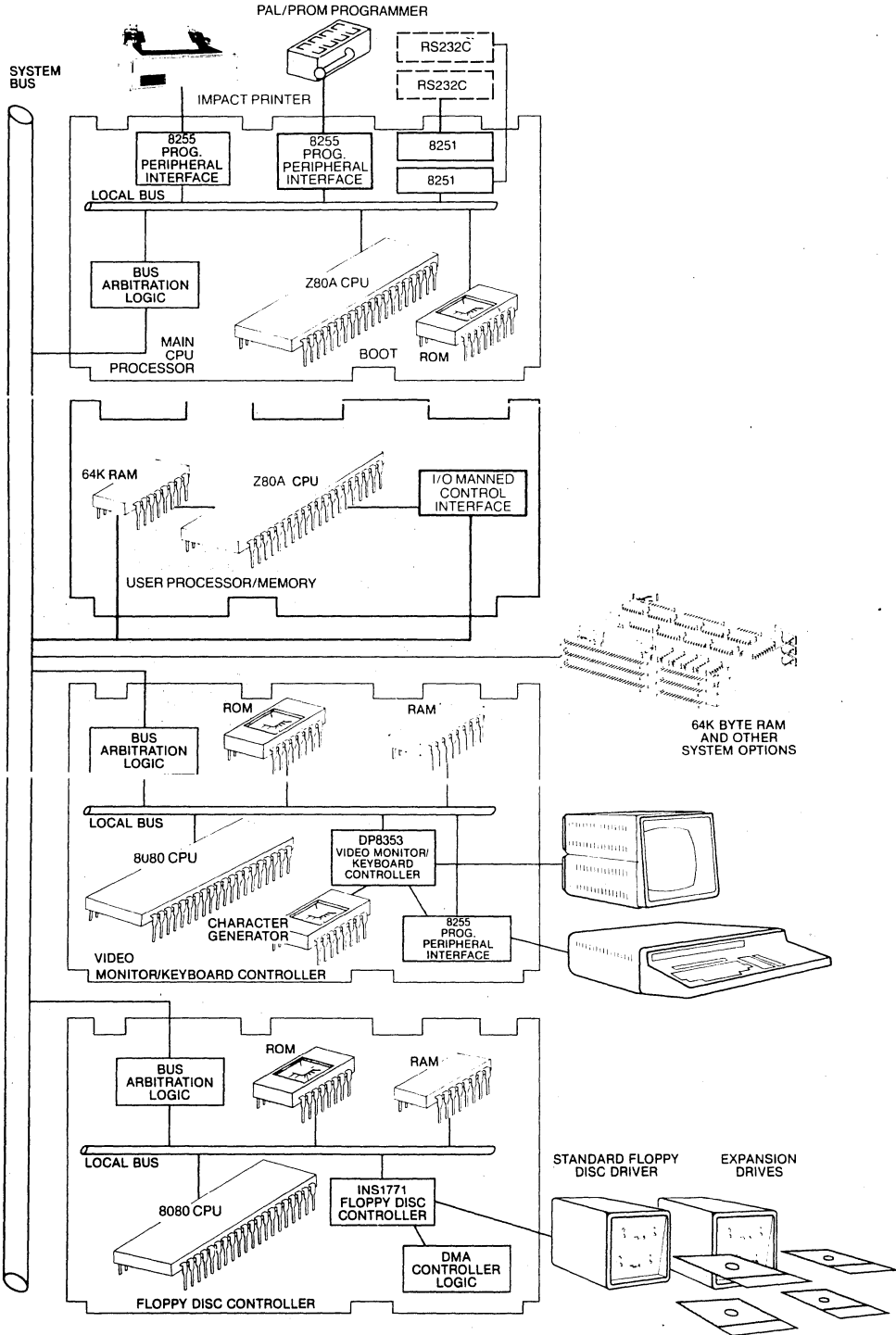
58 alphanumeric keys

### System Reset Boot Load Button

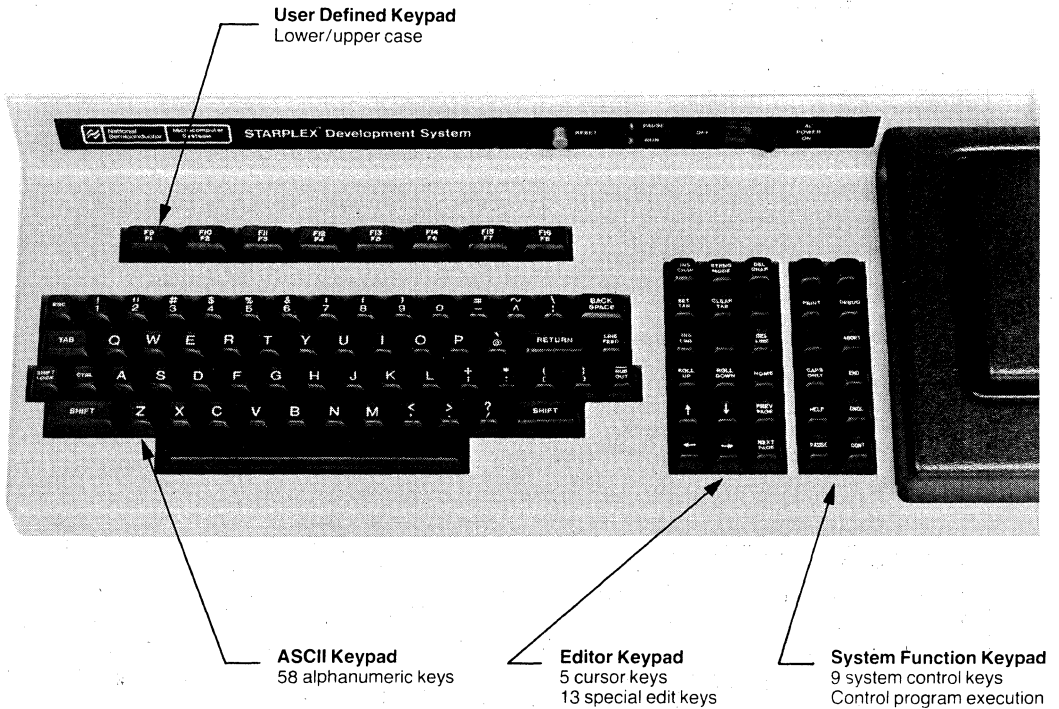
Powerful resident bootstrap has built-in  
 micro-diagnostics to check all system  
 facilities on initialization, then  
 automatically switch out of user  
 memory space



# STARPLEX II Multiprocessor System



# STARPLEX II Keyboard



## Standard Configuration

IN THE STANDARD CONFIGURATION, STARPLEX II provides a fully functioning turnkey system including the following features:

- CPU Master
- CPU Slave
- Bootstrap and diagnostic utility
- Two RS232C serial I/O ports
- Real time clock/calendar
- 128K bytes of mappable RAM
- Keyboard base
- Video monitor with 7 × 9 dot matrix and 1920 character display
- Dual floppy disk subsystem with double-density (1 mb) or double-sided double-density (2 mb) disk drives
- Debugger for diagnosing program execution
- Additional utilities for system maintenance
- Expansion slots for Integral ISE capability
- BASIC interpreter
- FORTRAN compiler
- Modular construction for versatility in operation
- Expansion capabilities to meet your growing requirements
- Complete operating system including an input/output system with an independent interface to user tasks
- File manager for comprehensive data storage and retrieval file creation, protection, deletion and attribute assignment with use of unique keyboard utility keys
- Screen oriented text editor for creating and editing source statements
- Macro assembler for assembling Z80 mnemonics and user-defined macros
- Linker for linking independent program modules into executable files
- PROM programming capability including interface board and universal software with PAL support

## Order Information

SPX-90/51	STARPLEX II Development System with 1 Megabyte Disk Storage (single-sided, double-density drives) (60 Hz)
SPX-90/61	STARPLEX II Development System with 2 Megabyte Disk Storage (double-sided, double-density drives) (60 Hz)

## Options

SPM-90-A02-1	PROM programming module for programming 2708 EPROMs
SPM-90-A02-2	PROM programming module for programming 2716 EPROMs

SPM-90-A06-1	STARPLEX II 1 Megabyte Dual Disk Expansion
SPM-90-A06-2	STARPLEX II 2 Megabyte Dual Disk Expansion
SPM-90-A08	In-System Emulator Module
SPM-90-A09-1	8080 Emulator Package
SPM-90-A09-2	8048 Emulator Package (includes upgrade kits that convert ISE 8048 to emulate 8049 and 8050)
SPM-90-A09-3	8070 Emulator Package (includes upgrade kits that convert ISE 8070 to emulate 8070 and 8073)
SPM-90-A10	Z80 Development Package
SPM-90-A13	Integral In-Systems Emulator Package
SPM-90-A13-3	8085 Emulator Package
SPM-90-A13-4	NSC800 (5V) Emulator Package
SPM-90-A13-7	Z80 Emulator Package
SPM-90-A15	COPS Emulator Package
SPM-90-A55	Impact Printer
SFW-90-A001	8048 Cross Assembler
SFW-90-A002	8070 Cross Assembler
SFW-90-A003	NSC800 Cross Assembler
SFW-90-A006	COPS Cross Assembler
SFW-90-A009	8080 Cross Assembler
SFW-90-A50	PL/M Compiler for 8080/8085
SFW-90-A60	PL/M Compiler for NSC800/Z80
SFW-90-A100	PAL/PROM Programming Software
SFW-90-A200	CP/M Operating System Software Package
SFW-90-A300	PASCAL Compiler for 8080/8085 <sup>1</sup>
SFW-90-A320	PASCAL Compiler for NSC800/Z80 <sup>2</sup>
AEE-90-A001	STARLINK — SPX/MDS220, 230 Link
AEE-90-A002	STARLINK — SPX/MDS800, 888 Link

**Note:** To order 50 Hz add the letter "E" to the order.

1. Not available at the time of this writing.
2. Available in December 1981.



<b>Documentation</b>		420306240-001	SPM-90-A13-3 8085 Integral ISE User's Manual
<b>STARPLEX II Development System</b>			
420306465-001	STARPLEX II System Hardware Reference Manual	420306241-001	SPM-90-A13-4 NSC800 (5V) Integral ISE User's Manual
420306383-001	STARPLEX II System Software Reference Manual	(See 1. below)	SPM-90-A13-7, Z80 Integral ISE User's Manual
420305788-001	STARPLEX II Macro Assembler Software User's Manual	420306254-001	SPM-90-A15 COPS ISE User's Manual
420305790-001	STARPLEX II FORTRAN Compiler Software User's Manual	<b>STARPLEX II Development System Software</b>	
420305791-001	STARPLEX II BASIC Interpreter Software User's Manual	420305789-001	SFW-90-A009 8085/8085 Cross Assembler Software User's Manual
420305804-001	BLC-8222 Double-Density Floppy Disk Controller Hardware Reference Manual	420306050-001	AEE-90-A001 STARLINK, STARPLEX II/MDS220/230 Software Manual
420305587-001	BLC-8228/8229 Video Monitor/Keyboard Controller Hardware Reference Manual	420306050-002	AEE-90-A002 STARLINK, STARPLEX II/MDS800/888 Software Manual
420305529-001	BLC-032/048/064 32/48/64K RAM Board Hardware Reference Manual	420306064-001	SFW-90-A001 8048 Family Cross Assembler User's Manual
<b>STARPLEX II Development System Options</b>			
420305653-001	SPM-90-A09-1 8080 ISE Target Board User's Manual	420306123-001	SFW-90-A002 8070 Family Cross Assembler User's Manual
420305869-001	SPM-90-A08 In-System Emulator Reference Manual	420306154-001	Z80 Cross Assembler Manual
420306065-001	SPM-90-A09-2 8048 ISE Target Board User's Manual	420306198-001	SFW-90-A003 NSC800 Cross Assembler User's Manual
420306132-001	SPM-90-A09-3 8070 ISE Target Board User's Manual	420306253-001	SFW-90-A006 COPS Cross Assembler User's Manual
420306155-001	SPM-90-A10 Z80 Development System Reference Manual	420306344-001	SFW-90-A200 CP/M Operating System User's Software Manual
420306183-001	SPM-90-A02 Universal PAL/PROM Programmer User's Manual	420306371-001	SFW-90-A50, SFW-90-A60 PLM80 Software Reference Manual

1. Not available at the time of this printing.

2. Available December 1981.

STARLINK, Quiklook, STARPLEX, STARPLEX II and ISE are trademarks of National Semiconductor Corp.

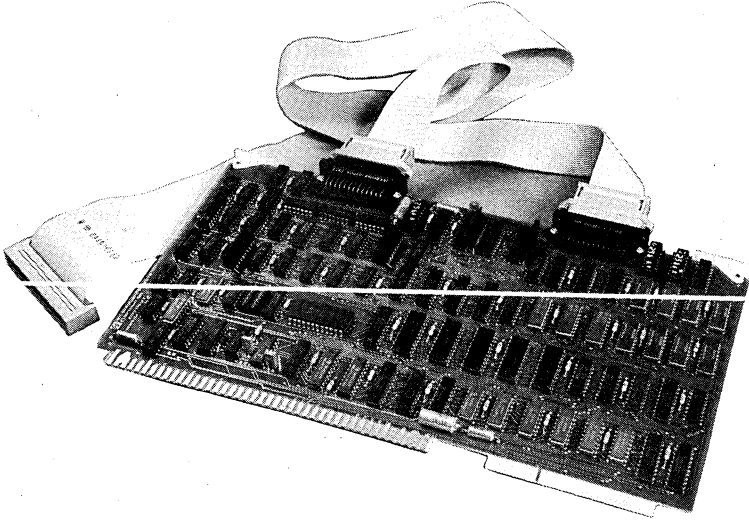
PAL is a registered trademark of Monolithic Memories.

Z80 is a registered trademark of Zilog.

COPS is a trademark of National Semiconductor Corp.

Information contained herein is intended to be a general product description and is subject to change. National does not assume any responsibility for use of any circuitry described, no circuitry patent licenses are implied, and National reserves the right, at any time without notice, to change said circuitry.

# **COPS™ In-System Emulator (ISE™) Package**



- **True Real-Time Emulation of the COP400 Family of Microcontrollers**
- **Plugs Directly into Any STARPLEX™/STARPLEX II™ Development System**
- **Compatible with the Required Optional COP400 Family Emulator Boards**
- **Easy to Use**
  - **Hardware**
    - Real-time trace of  $256 \times 20$ -bit instruction cycles
    - $4K \times 8$ -bit of Shared RAM Memory for rapid downloading of programs from STARPLEX/STARPLEX II peripherals
    - $1K \times 12$ -bit dump memory used in place of control firmware
    - External hardware breakpoint
  - Breakpoint timer in milliseconds
  - Fully compatible with a STARPLEX/STARPLEX II system bus
  - One target card handles entire series of microcontrollers and COP400 Emulator Boards
- **Software**
  - Software breakpoints
  - Lists user-specified registers when selected breakpoint is detected
  - Mnemonic modification of object code
  - Step-list-restart command
  - Dump routines for various COPS microcontroller chips

## **Product Overview**

The COP400 In-System Emulator (ISE) is designed for users with the STARPLEX/STARPLEX II Development System. Coupled with the power of STARPLEX/STARPLEX II, COP400 ISE is a very powerful tool available for developing and debugging COP400 family based microcontroller products. The COP400 ISE target board

plugs directly into any STARPLEX/STARPLEX II Development System and interfaces easily with any COP400 system. The designer has the capability of executing the target system program in real-time while collecting up to 256 instruction cycles of true real-time trace data. In addition, he can single step through his program and display the data from a 4K Shared Memory location.

## Functional Description

### Hardware

The COP400 ISE hardware consists of a printed circuit board (Target Board) which resides in the STARPLEX/STARPLEX II chassis. This target board interfaces via a flat ribbon cable to a required external emulator board which interfaces to the user's prototype system. This interface from the emulator board to the user's prototype system is accomplished through a COP400 pin-compatible plug — e.g., 20, 24 or 28 pin pin-compatible plugs, depending on the microcontroller chip. With the external COP400-E02, COP400-E02C, COP400-E04L, COP400-E24 or any other COP400 Emulator Board, a designer can perform emulation of the entire COP400 family of microcontrollers. They include:

- COP420, COP420L, COP420C
- COP421, COP421L, COP421C
- COP444L
- COP445L
- COP410L, COP410C
- COP411L, COP411C
- COP440, COP441, COP442
- COP2440, COP2441, COP2442

**Note:** "C" and "L" denote CMOS and Low-power versions respectively.

The COP400 ISE target board has  $4K \times 8$ -bit of Shared RAM Memory to allow rapid downloading of programs from STARPLEX/STARPLEX II peripherals. Also implemented on the target board is a single hardware breakpoint to allow the user to halt execution of the user program at a specified point in order to obtain information on the internal state of the COPS microcontroller device under emulation before resuming execution.

Also, on the target board is a  $1K \times 12$ -bit dump memory, used in place of a control firmware. The purpose of this dump memory is to allow different dump routines, contained on the main host driver diskette, to be entered in the dump memory for different microcontroller chips. Thus, the target board can be used for the entire series of microcontrollers.

On the Emulator Boards are two features that facilitate tracing. They are: 1) a "Trace Out" test point to help trigger oscilloscopes and logic analyzers, and 2) four user defined "external event" inputs into the Trace Logic circuitry to allow the user to define his own "events" for tracing.

### Software

The COP400 ISE software is a STARPLEX/STARPLEX II systems program which performs as the interface between the STARPLEX/STARPLEX II user and the COPS hardware system. The host driver, called COPMON™, allows the user to interrupt the flow of a program as it is being executed. The interruption is directly controlled by one of several events, all under user control. This interruption is called a "breakpoint." Possible conditions for a

breakpoint are "address," "next instruction," or any combination of two external events. COPMON can maintain a ten (10) level "condition" stack to aid easy debugging of large programs. In addition, COPMON can be specified to "break" only on the nth occurrence of a particular condition. A breakpoint timer allows COPMON to display the time in milliseconds between two "conditions."

COPMON also has one other primary function, "trace" control and display. The trace command allows the user to specify: 1) conditions that will initiate the trace and 2) how many steps prior to meeting that condition will be traced. The "Go" (see Command Summary below) command then arms the trace logic and executes the user's program. After a trace has been completed, the user may wish to examine the trace data by using a "TType" command or the user may wish to search for an address in the trace memory by using a "SEearch" command.

The COPS Cross-Assembler is also included with the COPS STARPLEX System Software package. It assembles COPS programs written with the STARPLEX Editor and stores them as object code load modules on the system diskette. There the load modules are accessible to the COPMON program which loads them into the Shared Memory on the COP400 ISE target board and executes them through the Emulator Board.

The third program included with the software package is called MASKTR™. MASKTR accepts final object code load modules prepared by the cross-assembler as input files and translates them into "Transmittal Files" which are stored on another diskette. The Transmittal Files each are in a format acceptable for National Semiconductor to prepare "hard" mask patterns from for custom ROM-based COP400 chip programs. A Transmittal File contains:

1. Name and phone number of the customer
2. Company name and address
3. Date
4. Chip number
5. Listing of option showing option number, option name, and option value
6. ROM data including addresses
7. Source, object, and transmittal file checksums.

### COPMON Console Command Summary

<b>ALter</b>	Alter Shared Memory
<b>AUtoprint</b>	Breakpoint printout control
<b>Breakpoint</b>	Breakpoint condition/occurrence control
<b>Clear</b>	Clears Breakpoint and Trace enables, and disables Timer
<b>CHip</b>	Selects Chip under emulation
<b>COmpare</b>	Compares Shared Memory to a disk file
<b>Deposit</b>	Deposit value into Shared Memory
<b>Find</b>	Searches Shared Memory for a specified value

<b>END</b>	Exit COPMON
<b>Go</b>	Begin Program Execution
<b>Help</b>	Prints out complete COPMON command summary for quick reference
<b>List</b>	Prints out the contents of Shared Memory
<b>LOad</b>	Loads Shared Memory from a disk file
<b>Modify</b>	Alters register contents of COPS chip under emulation
<b>Next</b>	Executes a single instruction but skips subroutines
<b>Put</b>	Alters Shared Memory mnemonically
<b>Reset</b>	Resets the COPS device
<b>Singlestep</b>	Executes a single instruction
<b>SAve</b>	Saves Shared Memory in a disk file
<b>SEarch</b>	Searches Trace memory for a specified address
<b>SET</b>	Set SIOMODE or STACKMODE Flags
<b>SHared Mem</b>	Enables Shared Memory operation
<b>STatus</b>	Prints out the emulation status
<b>Time</b>	Breakpoint timer control
<b>TYpe</b>	Prints out register contents of COPS chip under emulation
<b>TRace</b>	Set Trace Conditions
<b>UNassemble</b>	Display Instruction Mnemonics of the data in Shared Memory
<b>MASKTR Console Command Summary</b>	
<b>Abort</b>	Aborts the creation of a Transmittal File
<b>COmpany</b>	Prompts for Company Name and Address
<b>Date</b>	Prompts for Date
<b>Error</b>	Summarizes any option conflicts
<b>Finish</b>	Finishes the creation of the Transmittal File
<b>List</b>	Lists the Transmittal File
<b>Name</b>	Prompts for name/phone number of the person responsible for the program
<b>Option</b>	Prompts for the valid options
<b>Print</b>	Prints allowable options for chip specified
<b>Transmittal</b>	Load "Load Module" into memory

## Specifications

**Note:** The following specifications apply when the COPS ISE is configured with a standard COP400-E04L Emulator Board.

### Environmental

Operating Temperature	0°C to 40°C
Storage Temperature	-40°C to 85°C
Humidity (without condensation)	10% to 90%
Shock (Drop)	30g on 3 axis in shipping container

### Power (DC Characteristics — SPM-A15/SPM-90-A15 Power Consumption for Multiple Configurations)

Target Board	+5 VDC
Emulator Board	+5 VDC, -12 VDC

	Typical	Reasonable Worst Case
Target with Emulator and no PROMS	2.25A	3.20A
Target with Emulator with PROMS	2.50A	3.75A
Maximum User Supplied VCC	150mA	250mA

### Physical

Target Board	Length	6.75 inches
	Width	12.00 inches
	Depth	0.50 inches
Emulator Board	Length	6.50 inches
	Width	4.55 inches
	Depth	1.00 inches (with 4 - 0.50 inch nylon standoffs)

### Cables

Target/Emulator	Length	3 feet
	Material	50 × 28 AWG flat ribbon
	Termination	50-pin PCB edge RS232 male RS232 female
Emulator: User	Length	Approx. 1 foot
	Material	20, 24 or 28 × 28 AWG flat ribbon
Termination		20-pin IC plug both ends
		24-pin IC plug both ends
		28-pin IC plug both ends

**COPS Emulator/User Interface**

**COP411L – 20-Pin**

Pin	Signal
1	L4
2	VCC
3	L3
4	L2
5	L1
6	L0
7	SI
8	SO
9	SK
10	GND
11	L5
12	L6
13	L7
14	RESET/
15	CKI
16	D0
17	D1
18	G2
19	G1
20	G0

**COP 421/410/445 – 24-Pin**

Pin	Signal
1	GND
2	CKO
3	CKI
4	RESET/
5	L7
6	L6
7	L5
8	L4
9	VCC
10	L3
11	L2
12	L1
13	D0
14	D1
15	D2
16	D3
17	G3
18	G2
19	G1
20	G0
21	SK
22	SO
23	SI
24	L0

**COP420/444 – 28-Pin**

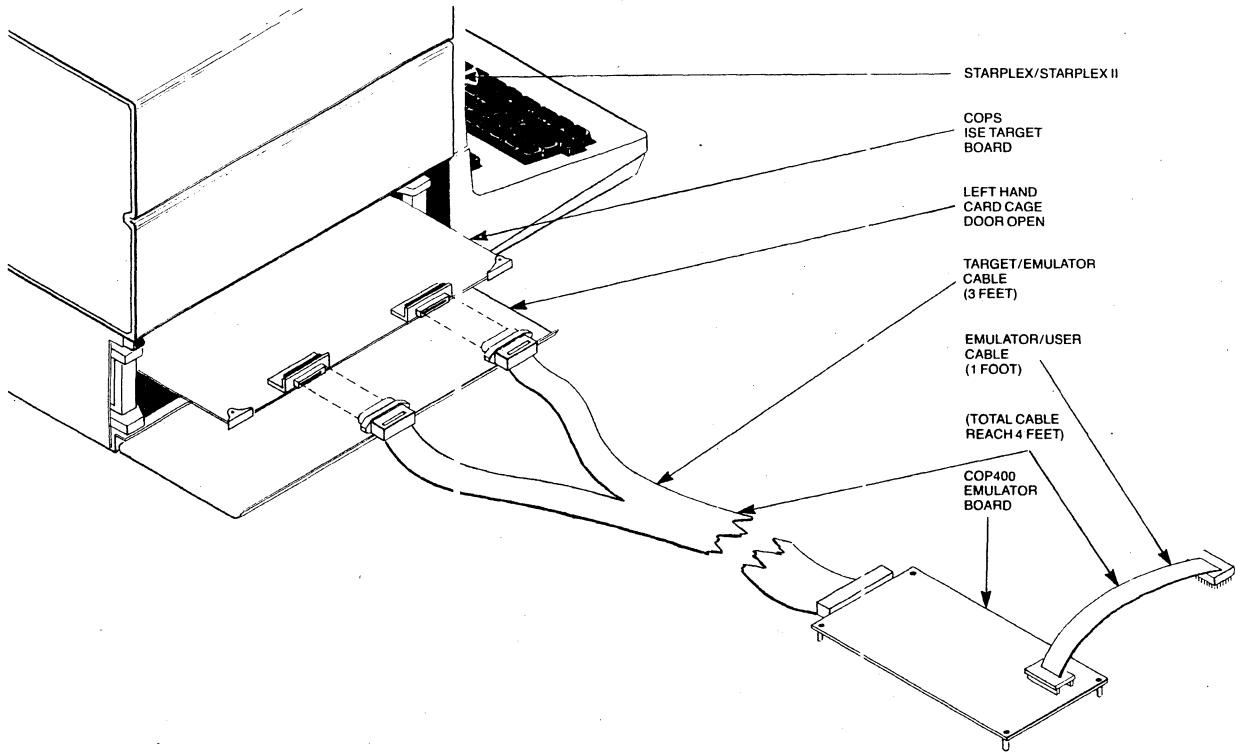
Pin	Signal
1	GND
2	CKO
3	CKI
4	RESET/
5	L7
6	L6
7	L5
8	L4
9	IN1
10	IN2
11	VCC
12	L3
13	L2
14	L1
15	D0
16	D1
17	D2
18	D3
19	G3
20	G2
21	G1
22	G0
23	IN3
24	IN0
25	SK
26	SO
27	SI
28	L0

**User Plug DC Characteristics Combined Specs For All Three Sockets**

Signal	Symbol	Parameter	Conditions	Value		Unit
				Min	Max	
L0 – L7	VOH	Voltage, Output High	IOH = 100 $\mu$ A IOL = 1.6mA	2.4		V
D0 – D3	VOL	Voltage, Output Low			0.4	V
G0 – G3	IOH	Current, Output High			-100	$\mu$ A
SO, SK	IOFF	Hi-z Output Leakage			+10	$\mu$ A
CKO	IOL	Current, Output Low			1.6	mA
L0 – L7	VIH	Voltage, Input High		2.0		V
CKI	VIL	Voltage, Input Low			0.8	V
SI						
IN0 – IN3	VIH	Voltage, Input High		.7VCC		V
G0 – G3	VIL	Voltage, Input Low			0.6	V
RESET/		Hysteresis		1.0		V

**Prerequisites**

Any STARPLEX/STARPLEX II Development System and a COP400-E02, COP400-E04L, COP400-E02C, COP400-E24 or any other COP400 Emulator Board.



**Installation of the COPS ISE Target Board and an Emulator Board**

**Order Information**

(Includes ISE Target Board, STARPLEX/STARPLEX II Emulator Cable, Cross Assembler, complete software and user's manuals, software to create a disk file for transmission of customer ROM patterns and device I/O options.)

For STARPLEX Development Systems:

SPM-A15 COPS In-System Emulator (ISE) Package

For STARPLEX II Development Systems:

SPM-90-A15 COPS In-System Emulator (ISE) Package

COP400 Family Emulator Boards:

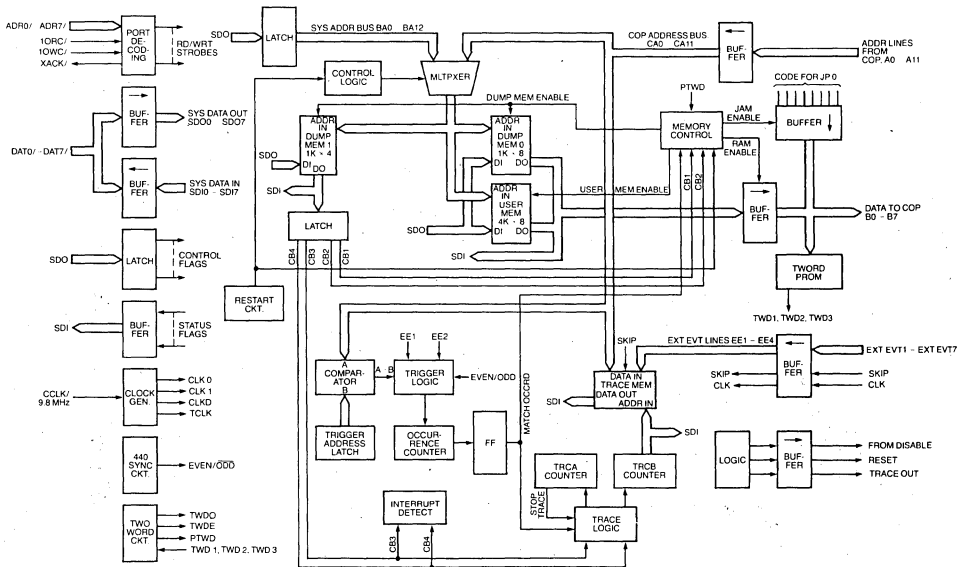
COP400-E02 402 Emulator Board

COP400-E02C  
COP400-E04L  
COP400-E24

CMOS Emulator Board  
404L Emulator Board  
440/2440 Emulator Board

**Documentation**

420305785-001 COP400 Microcontroller Family User's Manual  
420306469-001 COP400 In-System Emulator Card User's Manual  
420306253-001 COPS Cross-Assembler Software User's Manual  
420306254-001 COPS ISE Operator's Manual



**Target PWA Block Diagram**

# STARPLEX™

## COPS™

### Cross-Assembler

## User's Manual



Publication No. 420306253-001C  
Order No. 420306253-001  
August 1981



# Preface

This manual describes the COPS™ Cross-Assembler, a support program that allows a user to assemble a source program and generate object code on the STARPLEX™ Development System that executes on COPS microprocessor systems. Detailed information on formats and syntax is provided, but no attempt is made to teach assembly language programming to the STARPLEX system user.

The following manuals provide further information on the STARPLEX Development System:

- STARPLEX System Software Reference Manual  
Publication No. 420305788
- STARPLEX II™ Software Reference Manual  
Publication No. 420306383
- STARPLEX System Hardware Reference Manual  
Publication No. 420305789
- STARPLEX II Hardware Reference Manual  
Publication No. 420306465

The material presented in this manual is for information purpose only as specifications for both the COPS Cross-Assembler and STARPLEX System are subject to change without notice.

# Cross-Assembler Table of Contents

Section	Description	Page
<b>Chapter 1. Introduction and Overview</b>		
1.1	Introduction .....	8-168
1.2	The STARPLEX Development System .....	8-168
1.3	COPS Family Cross-Assembler Features .....	8-171
1.4	COPS Chip Overview .....	8-171
<b>Chapter 2. COPS Assembly Language</b>		
2.1	Introduction .....	8-174
2.2	Language Elements .....	8-174
2.3	Statement Fields .....	8-179
<b>Chapter 3. Instruction Set</b>		
3.1	Introduction .....	8-182
3.2	COP420 Series/COP444L Instruction Set .....	8-182
3.3	COP410LCOP411L Instruction Set .....	8-189
3.4	Arithmetic Instructions .....	8-189
3.5	Transfer of Control Instruction .....	8-190
3.6	Memory Reference Instructions .....	8-192
3.7	Register Reference Instructions .....	8-194
3.8	Test Instructions .....	8-196
<b>Chapter 4. Directives</b>		
4.1	Introduction .....	8-198
4.2	Directive Format .....	8-198
4.3	Definition Directives .....	8-198
4.4	Symbol Definition Directive .....	8-199
4.5	Assembler Control Directives .....	8-199
4.6	Repetition Directives .....	8-201
4.7	Conditional Assembly Directives .....	8-201
4.8	COPS Instruction Set Directives .....	8-202
<b>Chapter 5. Macros</b>		
5.1	Introduction .....	8-204
5.2	Macro Directives .....	8-204
5.3	Basic Macro Concepts .....	8-204
5.4	Macros and Subroutines .....	8-205
5.5	Defining a Macro .....	8-205
5.6	Calling a Macro .....	8-206
5.7	Using Parameters .....	8-206
5.8	Local Symbols .....	8-206
5.9	Conditional Expansion .....	8-207
5.10	Macro-Time Looping .....	8-207
5.11	Nested Macro Calls .....	8-207
5.12	Nested Macro Definitions .....	8-207

Section	Description	Page
<b>Chapter 6. Operating Instructions</b>		
6.1	Introduction .....	8-208
6.2	Invoking the Assembler .....	8-208
6.3	Object File Format .....	8-208
6.4	Listing Format .....	8-210
6.5	COPS Cross-Assembler Messages .....	8-210
<b>Chapter 7. Programming Techniques</b>		
7.1	Introduction .....	8-211
7.2	Program Memory Allocation .....	8-211
7.3	Data Memory Allocation and Manipulation .....	8-211
7.4	Subroutine Techniques .....	8-214
7.5	Timing Considerations .....	8-215
7.6	Programming Techniques for the COP421 Series, COP410L and COP411L .....	8-215
<b>Chapter 8. Sample Programs</b>		
8.1	Register Move Routines .....	8-217
8.2	BCD Arithmetic Routines .....	8-218
8.3	Simple Display Loop Routine .....	8-220
8.4	Interrupt Service Routine .....	8-221
8.5	Timekeeping Routine .....	8-222
8.6	String Search Routine .....	8-224
<b>Appendices</b>		
A	ASCII Character Set .....	8-225
B	Alphabetic Mnemonic Index of ASMCOP Instructions .....	8-226
C	Numeric Index of ASMCOP Instructions .....	8-227
D	Directive Summary .....	8-229
E	Programmer's Checklist .....	8-229
F	Positive Powers of Two .....	8-230
G	Negative Powers of Two .....	8-231
H	The Hexadecimal Number System .....	8-232
I	Hexadecimal and Decimal Integer Conversion .....	8-234
J	Hexadecimal and Decimal Fraction Conversion .....	8-235
K	Negative Hexadecimal Numbers .....	8-236
L	Program Listing Format .....	8-237
M	Load Module Format .....	8-238
N	Assembler Error Messages .....	8-239

Figure	Illustrations	Page
1-1	Illustration of the Process of Creating an Executable Program .....	8-170
1-2	Program Memory Map for COP420 .....	8-171
2-1	Relationship of Terms .....	8-176
2-2	Example of Symbol Usage .....	8-177
2-3	Sample Program Illustrating Fields .....	8-180
2-4	Label Field in a Source Program .....	8-180
2-5	Operation Field in a Source Program .....	8-181
2-6	Operand Field in a Source Program .....	8-181
2-7	Comment Fields in a Source Program .....	8-181
2-8	Source Program with Unaligned Fields .....	8-181
5-1	Statement Insertion .....	8-181
6-1	LM File Format .....	8-209
6-2	Title Record Format .....	8-209
6-3	Data Record Format .....	8-209
6-4	End Record Format .....	8-210
7-1	COP420 Data Memory Map .....	8-214
Table	Tables	Page
1-1	Enable Register Modes — Bits EN3 and EN0 .....	8-172
2-1	Number Representation .....	8-179
2-2	Arithmetic Operators .....	8-179
2-3	Logical Operators .....	8-179
2-4	Relational Operators .....	8-179
3-1	COP420 Series/COP444L Instruction .....	8-183
3-2	COP410L/COP411L Instruction Set .....	8-186
5-1	Macro Directives .....	C-201
7-1	Page to Hexadecimal Address Table .....	8-212

# Introduction and Overview

## 1.1 General Description

A cross-assembler is a program that executes on one type of computer and assembles source programs written in the assembly language of a different type of computer. Cross-assemblers produce object modules for execution (after a suitable transfer operation) on the second type of computer.

The COPS™ Cross-Assembler executes on the STARPLEX™ Development System and assembles COPS assembly language source programs into Load Modules. The Load Modules created are executable only on those systems which are COPS (or equivalent) microprocessor-based.

This manual describes the COPS Cross-Assembler as follows:

Chapter 1 (Introduction and Overview) introduces the cross-assembler, summarizes its characteristics/features, and describes COPS microprocessor features.

Chapter 2 (Assembly Language) describes the language elements: character set, number representation, character strings, instruction and directive formats, expressions, and relocation rules.

Chapter 3 (COPS Instructions) describes each instruction individually.

Chapter 4 (Assembler Directives) describes each directive individually.

Chapter 5 (Macros) gives detailed information on how to use the macro instructions.

Chapter 6 (Assembler Operation) details COPS Cross-Assembler operating instructions.

Chapter 7 (Programming Techniques) describes programming techniques that ease the task of writing COPS programs.

Chapter 8 (Sample Programs) shows some very useful sample programs that use the programming techniques described in Chapter 7.

Appendices A through I include the following information:

- A ASCII Character Set
- B Alphabetic Listing of Instructions
- C Numeric Listing of Instructions
- D Directive Summary
- E Programmer's Checklist
- F Positive Powers of Two
- G Negative Powers of Two
- H The Hexadecimal Number System
- I Hexadecimal and Decimal Integer Conversion
- J Negative Hexadecimal Numbers
- K Program Listing Format
- L Load Module Format
- M Assembler Error Messages

## 1.2 The STARPLEX Development System

### 1.2.1 In-System Emulation

The COPS In-System Emulator is a software development tool for users who wish to prototype systems involving one or more types of COPS microprocessors.

Once the In-System Emulator is installed in the STARPLEX Development System, the user needs only a single STARPLEX software-driver program to initiate the emulation process. The driver program is called by a single keystroke (STARPLEX only) or a typed command. A fill-in-the-blank menu appears on the CRT and prompts the user to select the microprocessor to be emulated.

### 1.2.2 STARPLEX Compatibility

The COPS Family Cross-Assembler (ASMCOP) is compatible with the STARPLEX Development System and its Operating System (OS) except for differences in directives. The COPS Family Cross-Assembler uses the same source line format as the 8080 macro-assembler, the 8070 Cross-Assembler, and the 8048 Cross-Assembler. The output object file of the COPS Family Cross-Assembler is compatible with that required by the In-System Emulator.

### 1.2.3 Theory of Operation

An assembly language is a symbolic programming language that uses mnemonic equivalents of machine instructions. Using a symbolic rather than binary language has the following important advantages:

- Mnemonic operation codes can be used to designate an operation.
- Data and instruction addresses can be assigned symbolic names which can be referenced by other instructions.
- The programmer may specify constant data in alphabetic, hexadecimal, octal, or decimal format, rather than binary format.
- Symbolic programs are easily modified because additional statements may be inserted into an existing statement sequence without any concern for the changing of address in the existing instructions.

The relationship between assembly language programming and the eventual execution of instructions by the processor is discussed in the rest of this section.

The processor initially examines the contents of the location numbered in the program counter (PC), and the PC is incremented *before* each instruction fetch, or before the execution of the operation. In this operation, the processor steps through a sequence of instructions called a program.

All numbers in the memory and the processor are represented in the binary system. If the programmer wishes to write his program in binary notation, the interface between the programmer and the machine can be relatively simple. However, writing in binary notation is slow and cumbersome. Productivity increases dramatically when the programmer is able to use a more congenial symbolic language and takes advantage of mnemonic features.

The conversion of the program from a symbolic to a binary representation is performed by the assembler program that translates the symbolic mnemonics into a binary machine-language program. This conversion is called the assembly process.

The assembly process begins with the symbolic source program which contains two basic types of statements: (1) symbolic machine instructions, and (2) directives. Assembly language instruction statements are symbolic representation of actual binary machine language instructions. Directives are assembly-dependent statements that control the assembly process and generate data in the assembled program.

Operands of machine language instructions represent storage locations, registers, immediate data, or constant values. A machine language instruction statement may be identified by assigning a label to it. The value of the label is the address of the assembled machine-language instruction.

Two outputs are generated as a result of running a program (programmer-generated statements) through the assembler program: (1) an object module (typically on diskette) consisting of actual machine language instructions and data corresponding to the source program statements, and (2) a program listing showing source statements side-by-side with the object code instructions created from the statements. Most programmers work with the program listing once it is available. An example of a STARPLEX™ program listing is shown in Chapter 2.

The object module (on diskette) is in a form suitable for use by the In-System Emulator™.

#### 1.2.4 Assembler Features

There is a one-to-one relationship between assembly language statements and machine instructions. A feature of assemblers is that the number of assembly language statements is the same as the number of machine instructions. In high level languages, this is not necessarily the case since nearly all single statements in a high level language are translated into many machine instructions.

The COPS™ Family Cross-Assembler is not the simple type of assembler described above. In any program written in COPS assembler language, there is likely to be a high ratio of machine instructions to program statements because of the extensive repertoire of directives in the COPS assembly language.

Directives direct the assembler program to perform certain operations during the assembly process. The directed operations include: (1) assisting the programmer in data and symbol definition, (2) checking and documenting the program, (3) controlling the assignment of storage addresses, (4) sectioning and including programs, and (5) controlling the assembler auxiliary functions to be performed by the assembler program. Operands of directives provide the information needed by the assembler program to perform the designated operation. Refer to Chapter 4 for detailed information on directives.

The feature of the COPS assembly language that provides the greatest flexibility and efficiency for the programmer is the MACRO directive. The programmer can define any sequence of instructions as a MACRO, and the assembler then inserts the entire sequence of instructions in response to a single-line directive in the source program. Refer to Chapter 5 for detailed information on MACROS.

The assembler has a number of attractive features for the programmer. The organizational facilities that are represented by the extensive directives make programs easier to write and understand once they are written.

*Modularity*—The MACRO-directive capability makes it easier for the user to write program segments and debug these segments before the complete program is written.

*Division of Labor*—Because of the modularity, different programmers can work on various sections of a large program.

*Library*—Debugged MACROS can be stored in a library for use in other programs as the need arises. An extensive library of MACROS and simple calling facility of the assembly language constitutes a rich language that is tailor-made by the user for his own purposes.

*Productivity*—Once a MACRO is written by one programmer, it may be used by another programmer. This represents a productivity advantage.

*Flexibility*—Because MACROS are small, separately identifiable segments of programs, each MACRO may be changed from time to time as the need arises. A large program may be updated in this fashion without much effort.

#### 1.2.5 Relationship of the COPS Family Cross-Assembler to Other Software

The COPS Family Cross-Assembler accepts source program files written in COPS assembly language and generates an absolute machine language program (load module). Normally, these source programs have been created using the STARPLEX editor program. Figure 1-1 illustrates the process of creating a COPS program.

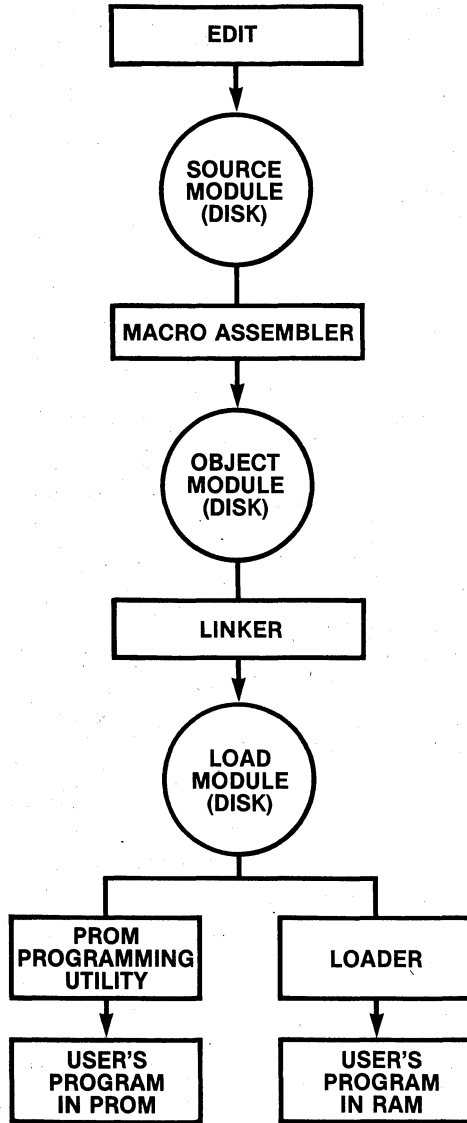


Figure 1-1. Illustration of the Process of Creating an Executable Program

The user normally debugs the program using the In-System Emulator™ and a corrected version of the program is created, using the COPST™ Family Cross-Assembler. The cycle repeats itself until the user is satisfied with the operation of the program. At this stage, the user may, depending on the nature of his development project, transfer the complete program to be run on another microcomputer system.

### 1.3 COPS Family Cross-Assembler Features

The ASMCOP Cross-Assembler produces an absolute object module from COPS assembly language programs. The directives and conventions used are upward compatible to the PDS assembler. The ASMCOP assembler is similar to the other STARPLEX Assemblers. However, there are some differences between ASMCOP and the others.

The COPS Family Cross-Assembler features include:

- 51 instructions
- Two-pass assembly
- Symbol table is built-in memory
- Input is accepted from test files on diskette
- Non-relocatable output is generated to diskette in the format described in Section 6.4
- Optional listing generated to CRT, line printer, or diskette
- Optional cross reference information within program listing
- Assembly directives for:
  - Data assignment
  - Control of location counter
  - Listing control
  - Conditional assembly
  - Repetition assembly
- Macro facilities

### 1.4 COPS Chip Overview

The section provides the programmer with a functional overview of the COPS chip. Information is presented at a level that provides a programmer with the background required to write efficient assembly language programs.

#### 1.4.1 Program Memory

Program memory consists of 512-byte ROM for COPS chips 410/411, a 1024-byte ROM for COPS chips 420/421, a 2048-byte ROM for COPS chips 444/445 and a 4096-byte ROM for COPS chips 440, 441, 442, and 2440, 2441, 2442 (see Figure 1-2). ROM words may be instructions, data or ROM address pointers. Due to the special characteristics associated with the JP and JSRP instructions, ROM must often be conceived of as organized into eight pages for COPS chips 410/411 (or 16 pages for COPS chips 420/421) of 64 words (bytes) each. Also, because of the unique operations performed by the LQID and JID instructions, ROM pages must often be thought of as organized into four consecutive blocks of ROM pages.

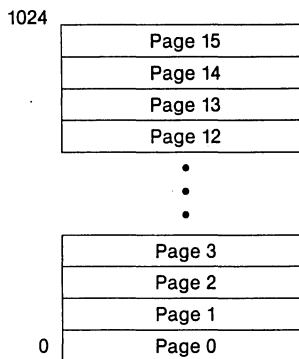


Figure 1-2. Program Memory Map for COP420

ROM addressing is accomplished by the P register. Its binary value selects one of the ROM 8-bit words (I7-I0) contained in ROM. The value of P is automatically incremented by 1 prior to the execution of the current instruction to point to the next sequential ROM location, unless the current instruction is a transfer of control instruction. In the latter case, P is loaded with the appropriate nonsequential value to implement the transfer of control operation performed by the instruction. It should be noted that P will automatically "roll over" to point to the next page of program memory. This feature has particular significance for instructions with paging restrictions, i.e., JP, JSRP, JID and LQID. Since P is incremented to roll over to the next ROM page prior to executing these instructions, they will be treated as residing on the next ROM page if they reside in the last word of a ROM page.

#### 1.4.2 Data Memory

Data memory consists of a RAM, organized as four 8-bit data registers for the 410/411 chips or eight 16-bit data registers for 444/445 chips. RAM addressing is implemented by a B register whose upper bits (Br) select one of the data registers and the lower bits (Bd) select one to 16 4-bit digits in the selected data register. While the 4-bit contents of the selected RAM digit (M) are usually loaded into or exchanged with the A register (accumulator), they may also be loaded into or from the Q latches or loaded from the L ports. RAM addressing may also be performed directly by the LDD and XAD instructions based upon the contents of the operand field of these instructions. The Bd register also serves as a source register for 4-bit data sent directly to the D outputs.

#### 1.4.3 Programmable Controls

**A Register.** The 4-bit A register (accumulator) is the source and destination register for most I/O arithmetic, logic and data memory access operations. It can also be used to load the Br and Bd portions of the B register, to load and input four bits of the 8-bit Q latch data, to input four bits of the 8-bit L I/O port data and to perform data exchanges with the SIO register.



**B Register.** The 6-bit (or 7-bit) RAM Address Register.

**C Register.** The 1-bit Carry register, in conjunction with the XAS instruction and the EN register, also serves to control the SK output. C can be output directly to SK or can enable SK to be a SYNC pulse, providing a clock each instruction cycle time.

**D Register.** The 4-bit Data Output Port. The D register provides four general-purpose outputs and is used as the destination register for the 4-bit contents of Bd.

**EN Register.** 4-bit Enable Register. The EN register is an internal 4-bit register loaded under program control by the LEI instruction. The state of each bit of this register selects or deselects the particular feature associated with each bit of the EN register (EN3-EN0).

1. The least significant bit of the enable register, EN0, selects the SIO register as either a 4-bit shift register or a 4-bit binary counter. With EN0 set, SIO is an asynchronous binary counter, decrementing its value by one upon each low-going pulse ("1" to "0") occurring on the SI input (count-down counter). Each pulse must be at least two instruction cycles wide. SK outputs the value of C upon execution of XAS and remains latched until the execution of another XAS instruction. The SO output is equal to the value of EN3. With EN0 reset, SIO is a serial shift register shifting left each instruction cycle time. The data present at SI goes into the least significant bit of SIO. SO can be enabled, via EN3, to output the most significant bit of SIO each cycle time. The SK output becomes a logic-controlled clock, providing a SYNC signal each instruction time. It will start outputting a SYNC pulse upon the execution of an XAS instruction with C = 1, stopping upon the execution of a subsequent XAS with C = 0.
2. With EN1 set, the COP420 IN1 input is enabled as an interrupt input. Immediately following an interrupt, EN1 is reset to disable further interrupts. Note that this interrupt feature associated with IN1 is available only on the COP420 and COP444L since only they have the IN inputs. Bit 1

(EN1) of the Enable Register is a "don't care" bit for those chips without the IN port. Setting or resetting this bit, via an LEI instruction, will have no effect on the operation of those chips. The chips that have an IN port are 420, 420L, 420C, and 444L.

3. With EN2 set, the L drivers are enabled to output the data in Q to the L I/O ports. Resetting EN2 disables the L drivers, placing the L I/O ports in a high-impedance input state. If the COP420 MICROBUS™ option is being used, EN2 does not affect the L drivers.
4. EN3, in conjunction with EN0, affects the SO output. With EN0 set (binary counter option selected), SO will output the value loaded into EN3. With EN0 reset (serial shift register option selected), setting EN3 enables SO as the output of the SIO shift register, outputting serial shifted data each instruction time. Resetting EN3 with the serial shift register option selected disables SO as the shift register output; data continues to be shifted through SIO and can be exchanged with A via an XAS instruction, but SO remains reset to "0". Table 1-1 provides a summary of the options and features associated with EN3 and EN0.

**G Register.** The 4-bit register to latch data for G I/O port. The G register contents are output to four general-purpose bidirectional I/O ports. The COP420 G0 pin may be mask-programmed as a "ready" output for MICROBUS applications. As with the IN3-IN1 COP420 MICROBUS options discussed below, this G0 MICROBUS option is not available for those chips lacking the IN ports.

**IL Latches.** Two 1-bit latches associated with the IN (3) or IN (0) inputs.

**IN 4-bits Input Port.** Four general-purpose inputs, IN3-IN0, are provided for the COP420. IN1, IN2, and IN3 may be selected, by a mask-programmable option, as Read Strobe, Chip Select and Write Strobe inputs, respectively, for use in MICROBUS applications.

**Table 1-1. Protection Levels and Safeguard Provisions**

EN3	EN0	SIO	SI	SO	SK after XAS
0	0	Shift Register	Input to Shift Register	0	If C = 1, SK = SYNC If C = 0, SK = 0
1	0	Shift Register	Input to Shift Register	Serial Out	If C = 1, SK = SYNC If C = 0, SK = 0
0	1	Binary Counter	Input to Binary Counter		If C = 1, SK = 1 If C = 0, SK = 0
1	1	Binary Counter	Input to Binary Counter		If C = 1, SK = 1 If C = 0, SK = 0

The COP421 does not contain the IN3-IN0 inputs and, therefore, must use the four bidirectional G I/O ports or eight bidirectional L I/O ports as input pins to the device. Also, due to its lack of the IN inputs, direct use of National's MICROBUS™ is inappropriate.

**L Register.** The 8-bit TRI-STATE® I/O port. The eight L drivers, when enabled, output the contents of latched Q data to the L I/O ports. Also, the contents of L may be read directly into A and M. As explained above, the COP420 MICROBUS option allows L I/O port data to be latched into the Q register. L I/O ports can be directly connected to the segments of a multiplexed LED display (using the TRI-STATE LED Direct Drive output configuration option) with Q data being outputted to the Sa-Sg and decimal point segments of the display.

**M Register.** The 4-bit contents of RAM memory pointed to by the B register.

**PC Register.** The 10- (9-, 11-, or 12-) bit ROM address register (program counter).

**Q Register.** The 8-bit register to latch data for L I/O port. The Q register is an internal, latched, 8-bit register, used to hold data loaded to or from M and A, as well as 8-bit program data from ROM. Its contents are output to the L I/O ports when the L drivers are enabled under program control (via an LEI instruction).

The COP420 may use the MICROBUS option to write L I/O port data into Q upon the occurrence of a WS pulse from the host CPU.

**SA, SB, SC Registers.** The 10- (11- or 9-) bit subroutine stack registers. Three levels of subroutine are implemented by the 10-bit subroutine stack registers, SA, SB, and SC, providing a last-in, first-out (LIFO) hardware subroutine stack.

**SIO Register.** The 4-bit shift register and counter. The SIO register functions as a 4-bit serial-in/serial-out register or as a binary counter, depending on the contents of the EN register. Its contents can be exchanged with A, allowing it to input or output a continuous serial data stream. SIO may also be used to provide additional parallel I/O when used as a shift register with its input or output connected to external serial-in/parallel-out shift registers.

**SK.** Logic controlled clock output. The 10-bit time-base counter divides the instruction cycle frequency by 1,024, providing a pulse upon overflow. The COP420 SKT instruction tests for the occurrence of this pulse, allowing the programmer to rely on this internal time-base rather than external inputs (e.g., 50/60 Hz signals) to implement "real-time" routines.

# COPS™ Assembly Language

## 2.1 Introduction

This chapter describes the following elements of the COPS Assembly Language:

- Character Set
- Delimiters
- Number Representation
- Character Strings
- Symbols
- Instruction Format
- Expression and Operators

All aspects of the ASMCOP assembler are compatible with the PDS COPS assembler except for some differences in the macros.

## 2.2 Language Elements

Input to the assembler consists of a sequence of characters combined to form assembly language elements. These elements include the symbols, instruction mnemonics, constants, and expressions that make up the individual program elements of a source program.

## 2.2.1 Character Set

Valid ASCII letters, numbers, and special characters are the same as described for all STARPLEX™ assemblers; however, the uses of some characters are different.

Assembly language source statements are written using the following letters, numbers, and special characters:

- Upper and lower case letters A through Z of the English alphabet
- Numbers 0 through 9

Character	Name
CR	Carriage return
LF	Line feed
FF	Form feed
HT	Horizontal tab
Blank	Blank or Space

- The following printable characters:

Character	Name	STARPLEX COPS	Other STARPLEX ASMs
+	Plus Sign	Addition	Addition
-	Minus Sign	Subtraction	Subtraction
*	Asterisk	Multiplication	Multiplication
/	Slash	Division	Division
\	Backslash*		
,	Comma	Delimits operands, para	Delimits operands
.	Period	Program counter	Can be in symbol name
;	Semicolon	Delimits comments	Delimits comments
:	Colon	Delimits label	Delimits label
'	Single Quote	Delimits strings	Delimits strings
"	Double Quote	Delimits strings	Delimits strings
?	Question Mark	Can be in symbol name	Can be in symbol name
!	Exclamation Mark	Logical OR	Escape character
&	Ampersand	Logical AND	Concatenation
\$	Dollar Sign	Begins local symbol	Program counter
@	At Sign	Escape character	Can be in symbol name
(	Left Parenthesis	Delimits expressions	Delimits expressions
)	Right Parenthesis	Delimits expressions	Delimits expressions
<	Left Angle Bracket	Less than, greater than	Macro parameter delimit
>	Right Angle Bracket	Less than, greater than	Macro parameter delimit
[	Left Square Bracket*		
]	Right Square Bracket*		
{	Left Bracket	Delimits macro parameters	No special meaning
}	Right Bracket	Delimits macro parameters	No special meaning
#	Pound Sign*		
%	Percent Sign	Logical NOT	No special meaning
=	Equal Sign	Assignment, relational	No special meaning
^	Grav Accent*		
~	Caesura*		
~	Tilde*		
↑	Up Arrow	Concatenation	No special meaning
_	Underscore	Can be in symbol name	Can be in symbol name

**Notes:**

1. Those characters above listed with an asterisk (\*) are legal characters only within comment statements.
2. Except in strings, lower case letters are equivalent to upper case, i.e., 'a' is the same as 'A'.
3. The null character (ASCII zero) is ignored on input.

### 2.2.2 Delimiters

In an assembly language program, some of the special characters function as delimiters. Delimiters define the end of a field or the end of a source statement. The following list defines the delimiters recognized by the assembler.

Character	Meaning	Use
blank	One or more blanks	Field separator or symbol terminator
HT	Horizontal Tab	Field separator or symbol terminator
,	Comma	Separates operands in the operand field
'...'	Single Quotes	Delimits a character string
"..."	Double Quotes	Delimits a character string
(...)	Parentheses	Delimits an expression
;	Semicolon	Delimits a comment field
[...]	Brackets	Delimits macro parameters
:	Colon	Delimits symbols used as labels
CR	RETURN Key	Ends an input statement/line

### 2.2.3 Number Representation

In COPS assembly language source programs, numbers may be specified in decimal, hexadecimal, octal, or binary representation. A one-letter terminator determines the representation as indicated below.

Representation	Terminator	Example
Binary	B	11010010B
Octal	O or Q	2763O or 2763Q
Decimal	D (or none)	2398D or 2398
Hexadecimal	H	0B52H

#### Notes:

- If a number begins with X' or a leading zero, then it is assumed to be hexadecimal.
- If a number does not begin with a leading zero or X' and no prefix or terminator is used, then the number is assumed to be decimal.
- Signed integers between -32768 and +32767, inclusive, can be specified using any of the above representation. This range is the maximum representable in 16 bits or two bytes.
- The digits in a number must agree with the specified base. For example, octal numbers may contain only the digits 0 through 7. A hexadecimal number may contain digits 0 through 9 as well as letters A through F.

### 2.2.4 Current Location Counter

A period "." as an operand is interpreted as the current value of the location counter at the time the instruction is assembled.

### 2.2.5 Evaluation of Expressions

An expression may contain combinations of symbols, operations and numbers. All expression values are evaluated to 16 bits. If a number is too large for 16 bits, it is evaluated modulo 65,536. Furthermore, if an expression has an 8-bit operand, then the result of the expression must be in the range 0 to 255, or in two's complement, the range -127 to +128.

The following operators are supported within expressions:

Arithmetic Operators	Description	Corresponding STARPLEX Operators
+	Unary or binary addition	+
-	Unary or binary subtraction	-
*	Multiplication	*
/	Division (remainder is discarded)	/

Logical Operators	Description	Corresponding STARPLEX Operators
%	Logical one's complement	NOT
&	Logical AND	AND
!	Logical OR	OR
H	Isolate high order byte	HIGH
L	Isolate low order byte	LOW

Relational Operators	Description	Corresponding STARPLEX Operators
=	Equal	EQ
<	Less Than	LT
>	Greater Than	GT

All operators except H and L result in a 16-bit value. Relational operations always result in the value -1 if true and 0 if false. All logical operations are performed using unsigned arithmetic.

### 2.2.6 Operator Precedence

Expressions are evaluated left to right. Operators with higher precedence are evaluated before other operators that immediately precede or follow them. When two operators have equal precedence, the left-most is evaluated first.

Parentheses can be used to override normal rules of precedence. The part of an expression enclosed in parentheses is evaluated first. If parentheses are nested, the innermost are evaluated first. Operator precedence is the same as in all STARPLEX™ assemblers. That order is (from highest priority to lowest):

- Parenthesized expressions
- H, L
- Multiplication/Division: \*, /
- Addition/Subtraction: +, - (unary and binary)
- Relational Operators: <, =, >
- % (Logical NOT)
- & (Logical AND)
- ! (Logical OR)

The relational and logical operators must be separated from their operands by at least one blank.

### 2.2.7 Terms

The relationship of terms is shown in Figure 2-1. The various types of terms are described in the following paragraphs.

### 2.2.8 Symbols

All symbols must conform to the following rules:

- One to six characters in length (the actual length may be up to 32 characters, but only the first six characters will be used).
- The first character must be a letter, dollar sign "\$", or a question mark "?". The symbol name may not begin with period because this indicates a directive name or the program counter. The symbol may not contain an "@" because this has special meaning in macros. If the first letter is a dollar sign, the symbol is assumed to be local.

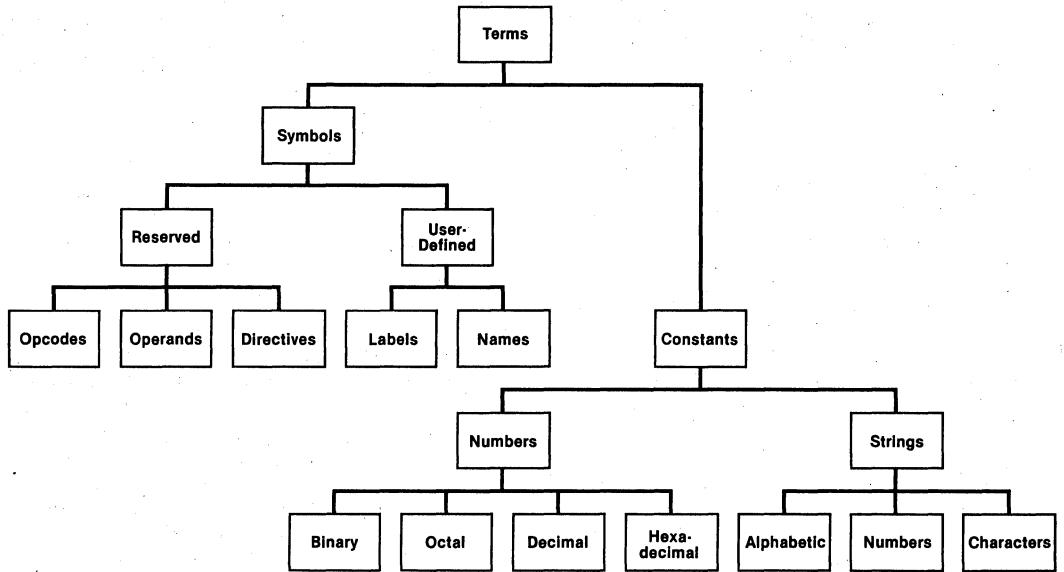


Figure 2-1. Relationship of Terms

3. Any remaining characters may be letters, digits, dollar sign "\$", question mark "?", period ".", or underscore "\_".
4. Symbol names may not be the single letter "L" or the single letter "H".

Symbolic representation of elements is superior to numeric representation for the following reasons:

1. You can give meaningful names to the elements of a program.
2. You can debug a program more easily, because the symbols are referenced in the symbol table at the end of the program.
3. You can maintain a program more easily, because you can change a symbolic value in one place and its value will be changed throughout the program.

Symbols are used in the label field of an instruction to identify the instruction and to represent its address. Symbols may be used for other purposes, such as the symbolic representation of a memory address, data constant, or register. Symbols used in this way must be defined before they are used. The assembler regards symbols as being reserved or user-defined.

*Defining a User Symbol.* No matter how the symbol is used, it must be defined. A symbol is defined when the assembler knows what value the symbol represents. There are two ways of defining a symbol. The symbol is assigned the current value of the location counter when it appears in the label field of an instruction, or it may be assigned some other value through the use of the SET or "=" directive. A symbol may not appear in the label field more than once in a program, because this would cause the assembler to try to redefine an already defined label. The assembler will not do this and it flags the second appearance of a label as an error.

*Examples of Symbol Usage.* Figure 2-2 shows a number of symbols used in a source program. Notice that symbols may appear in the label field or in the operand field of an instruction.

*Reserved, User-Defined, and Assembler-Generated Symbols.* Reserved symbols are symbols that have special meaning to the assembler and therefore cannot appear as user-defined symbols. The mnemonic names for the instructions and the directives are all reserved symbols.

*Location Counter.* The period refers to the current location counter. The location counter contains the address where the current instruction or data will be assembled.

Symbols	MULT:	LBI	0,13	
		JSR	CLR	
	MULT1:	LBI	2,0	
		JSR	TMZERO	Symbols
		JP	NOTZ	
		JSR	RSHR0	
		JSR	RSHR2	
		LBI	0,13	
		LD		
		AISC	3	
		JP	. +2	
		RET		

Figure 2-2. Example of Symbol Usage

## 2.2.9 Constants

A constant is a self-defining language element. Unlike a symbol, the value of a constant is its own "face" value and does not vary; the assembler does not assign a value to the term, but derives the value from the term.

Constants are used to specify immediate data, addresses, registers, and input/output information to the assembler. Five types of constants are available: binary, octal, decimal, hexadecimal, and character (or string). Constants are followed by a one character suffix that indicates the base. Numbers without a suffix are assumed to be decimal.

*Binary Constants.* A binary constant consists of 1 to 16 ones or zeros, followed by the letter "B". If there are less than 16 digits, leading zeros are assumed. The first digit of a binary constant cannot be a zero because hexadecimal constants begin with a zero.

Examples:

Valid	Invalid	Reason Invalid
10101B	1100100	No 0 after the last digit
111B	101 00B	Invalid character (the blank)
1B	01010101010101010101B	Too many digits
101111100B	101020B	Invalid character (the two)
11B	01B	Invalid starting number (the zero)

*Octal Constants.* An octal constant consists of a string of one to six digits followed by the letter "O" or the letter "Q". The octal digits are the numbers 0 through 7. Octal numbers in the range of 0 to 177777 are valid.

Examples:

Valid	Invalid	Reason Invalid
123456O	1234567	No 0 after the last digit
111222O	101 015O	Invalid character (the blank)
7Q	723455374O	Too many digits
7777Q	2345678O	Invalid character (the eight)

*Decimal Constants.* A decimal constant consists of one to five numeric characters. Optionally, decimal constants may be followed by the letter D. The value specified is right-justified. That means a 12 is equivalent to writing 00012 and not 12000. For 8-bit data, the value range is 0-255 for an unsigned decimal integer and +127 to -128 for a signed decimal integer. For 16-bit data the value range of a decimal constant is 0-65,535 for an unsigned decimal integer and +32,767 to -32,768 for a signed decimal integer. It should be noted that having signed and unsigned data is just a coding convenience made available because some instructions treat data as signed values and other treat data as unsigned values. For example, in 8-bit data, -1 and 255 both convert to the hexadecimal number FF.

You may wonder why the range of positive values for a signed number is one less than the range of negative values. Logically it would seem that if the most significant bit of binary number is the sign bit and the remaining bits specify the number, then the range for both positive and negative numbers would be the same. As it turns out, they are. The reason the positive numbers appear to be one less than the negative numbers is that zero is considered positive. This differs from mathematics where zero is considered neither positive or negative; but because most computers work on two's complement arithmetic, zero must be considered positive.

Examples:

Valid	Invalid	Reason Invalid
12	123456	Too many digits
-123	123-	Invalid character (the minus)
12345	12.34	Invalid character (the period)
+1234	12 34	Invalid character (the blank)
1234D	99999	Number outside allowed range

**Hexadecimal Constants.** A hexadecimal constant consists of one to four hexadecimal digits (0-9 and A-F) preceded by an "X" or a zero, or followed by an "H". If the first character is a zero, up to five hexadecimal digits may be specified.

Examples:

Valid	Invalid	Reason Invalid
X'1234	1234	No X' after the last digit, this is interpreted as a decimal number
0FFFF	FFFF	First character 0 or X' to indicate hexadecimal
0120	12 E	Invalid character (the blank)
357H	357	No "H" after the last digit, this is interpreted as a decimal number

**String Constants.** A string is a series of printable ASCII characters delimited by single or double quotes. Quotes may be part of a string by using two quote marks. For example, 'AB' 'C' represents the string AB'C.

Examples:

Valid	Invalid	Reason Invalid
'VALID'	INVALID	No single quotes
"valid"	'invalid	No quote following
'it's ok'	"it's not"	Two single quotes required to define a quote within a string

## 2.2.10 Expressions

An expression is an assembly language element that represents a value. It may consist of a single term or a combination of terms separated by arithmetic, relational, and/or logical operators. A term may be a valid symbolic reference, a self-defining constant, or a general constant. The result of the expression evaluation is an 8- or 16-bit value.

All of the operand types previously discussed can be combined by operators to form an expression. In fact, the example given for the location counter (. +10) is an expression that combines the location counter with the decimal number 10.

### 2.2.10.1 Operands of Expressions

**Symbols.** If a symbol is used as a label, its value (location) is the value of the location counter immediately before the corresponding source line is assembled.

Symbols can also be defined using the .SET directive or the '=' operator. Once a symbol has been defined, all references to that symbol are replaced with the corresponding address value. All values of symbols in COPS programs are absolute, not relocatable.

The COPS assembler allows local symbols. The following rules apply to the assignment and use of local symbols:

1. Local symbols can only be defined by using them as labels.
2. A dollar sign as the first character in the symbol name defines a local symbol.
3. Local regions are delimited by the .LOCAL directive.
4. A local symbol name must be unique in the first four characters, not including the dollar sign.
5. Local symbols defined in a local region are accessible only within that region of the program.

**Numbers.** All numbers are evaluated using 16-bit unsigned arithmetic. All numbers are evaluated modulo 65,536. This is equivalent to two's complement signed numbers for expression evaluation.

**Strings.** Each character in a string is evaluated as a byte whose value is the ASCII value of that character.

### 2.2.10.2 Arithmetic Operations

When discussing arithmetic operations, we must distinguish between assembly-time expression evaluation and program execution arithmetic. The numbers involved are represented identically in both cases, but program execution arithmetic has much more flexibility than assembly-time expression evaluation in determining the range of numbers, internal notation, and whether numbers are considered signed or unsigned. The characteristics of both modes of arithmetic are summarized in Table 2-1.

### 2.2.10.3 Permissible Range of Numbers

Numbers can range from 0 through 65,535 (0FFFFH). Numbers that are outside this range are evaluated "modulo" 64k (k = 1024). So, a number greater than 64k is divided by 64k and the remainder is substituted for the original number.

**Table 2-1. Number Representation**

Number Characteristic	Assembly-Time Expression Evaluation	Program Execution Arithmetic
Base representation	Binary, octal, decimal or hexadecimal	Any base
Range	0-65,535	User controlled
Evaluates to:	16 bits	User interpretation
Internal notation	Two's complement	Two's complement
Signed/unsigned arithmetic	Unsigned	Unsigned unless user manipulates

**2.2.10.4 Two's Complement Arithmetic**

In two's complement notation, negative numbers are formed by complementing all the bits in a number and adding a binary one to the result.

There is no subtraction instruction in the COPS instruction set. Subtraction is performed by taking the two's complement of the number to be subtracted and adding it to a second number (the minuend).

The CASC instruction performs a one's complement of the accumulator. To get a two's complement, you must complement and then add one to the result.

When a number is interpreted as a signed, two's complement number, the low-order bits are interpreted as the magnitude of the number and the high-order bit is interpreted as the sign. The range of a signed, two's complement number is -32,768 through +32,767 for 16 bits and -128 through +127 for eight bits.

When a 16-bit value is interpreted as an unsigned, two's complement number, it is considered to be positive and in the range of 0 through 65,535. An 8-bit value is in the range of 0 through 255.

All expression evaluation performed by the assembler assumes unsigned, two's complement numbers. Execution-time arithmetic also assumes unsigned, two's complement notation.

**2.2.10.5 Assembly-Time Expression Evaluation**

An expression is a combination of constants, symbols, and operators. Operators can be arithmetic, relational, and logical or specially-defined operators. Any symbol appearing in an expression must have been previously defined. All expression values are evaluated to 16-bits. If a number is too large for 16-bits, it is evaluated modulo 65,536. Furthermore, if an expression has an 8-bit operand, then the result of the expression must be in the range 0 to 255, or in two's complement the range -128 to +127.

All operators except H and L result in a 16-bit value. Relational operations always result in the value -1 if true and 0 if false. All logical operations are performed using unsigned arithmetic.

**2.2.10.6 Operators**

The assembler recognizes the following groups of assembly-time operators:

- Arithmetic
- Logical
- Relational

Table 2-2 gives the legal arithmetic operators.

**Table 2-2. Arithmetic Operators**

Operator	Meaning
+	Unary or binary addition
-	Unary or binary subtraction
*	Multiplication
/	Division. Remainder is discarded

Examples:

The following expressions generate an ASCII A:

```
5 + 30 * 2
(25/5) + 30 * 2
5 + (-30 * -2)
```

Table 2-3 gives the legal logical operators.

**Table 2-3. Logical Operators**

Operator	Meaning
%	Logical one's complement
&	Logical AND
!	Logical OR
H	Isolate high order byte
L	Isolate low order byte

Table 2-4 gives the legal relational operators.

**Table 2-4. Relational Operators**

Operator	Meaning
EQ	Equal
NE	Not equal
<	Less Than
>	Greater Than

The relational operators give a TRUE/FALSE result. If the evaluation of the relationship is TRUE, operations are based strictly on magnitude comparison of bit values. Therefore, a two's complement negative number has a greater value than a two's complement positive number, because a positive number always has a zero in its high-order bit position.

**2.3 Statement Fields**

Assembly language source lines consist of up to 131 ASCII characters. Source statements are divided into the following four fields:



	Name/Label Field	Opcode Field	Operand Field	Comment Field
01	000	00		
02	001	12		;WE MUST DO WHAT WE MUST DO
03	002	00	CLRRAM:	;CLEAR ALL RAM
04	003	04	CLR:	
05	004	C2	XIS	
06	005	12	JP	CLR
07	006	5D	XABR	
08	007	C1	AISC	13
09	008	32	JP	CLRRAM
10	009	4F	RC	
11	00A	3368	XAS	;TURN OFF SK
			LEI	8 ;ENABLE SHIFT REG

Figure 2-3. Sample Program Illustrating Fields

- Label/Name Field (optional)
- Opcode Field (mandatory)
- Operand Field (usually required)
- Comment Field (optional)

Spaces and tabs between fields and before the first field are allowed. An input line is terminated by a carriage return, and has the following general format:

[Label] Opcode [Operand,Operand] [;Comment]

The sample program shown in Figure 2-3 has the four fields delineated. However, since the COPS assembler accepts "free-form" statements, the programmer may disregard field boundaries. For clarity and readability, use of aligned boundaries, whenever possible, is highly recommended.

Following is an explanation of each field.

### 2.3.1 Label/Name Field

Labels are always optional. An instruction label is a symbol name whose value becomes the location where the instruction is assembled. A label may contain one to six alphanumeric characters, but the first character must be alphabetic. Alphanumeric characters include the letters of the alphabet and the decimal digits 0 through 9. The label name must be terminated with a colon (:). A symbol used as a label can be defined (appear in the label/name field) only once in your program.

A name is required for the .SET and "=" directives. Names follow the same coding rules as labels, except that they are terminated with a blank rather than a colon.

Figure 2-4 shows an example of labels in a source program.

Label Field

CLRRAM:	CLRA			;WE MUST DO WHAT WE
	XABR			;MUST DO
CLR:	CLRA			;CLEAR ALL RAM
	XIS			
	JP	CLR		
	XABR			
	AISC	13		
	JP	CLRRAM		
	RC			
	XAS			;TURN OFF SK
	LEI	8		;ENABLE SHIFT REG

Figure 2-4. Label Field in a Source Program

The following are some examples of the label field:

Valid	Invalid	Reason Invalid
LABEL:	123:	Begins with a decimal digit
F123:	LABEL	Not followed by a colon
WHERE:	ADD:	ADD is a reserved word

Since labels serve as instruction addresses, they cannot be duplicated. For example, the sequence:

```

HERE:   JMP   THERE
        .
        .
        .
THERE:  LDD   REG
        .
        .
        .
THERE:  SKMBZ 3
    
```

is ambiguous. The assembler cannot determine which address is to be referenced by the JMP instruction.

A label may appear by itself in a statement, in which case, it refers to the next instruction or data byte. For example, the following sequence is valid:

```

LABEL1:
LABEL2:  LDD   REG
        .
        .
        .
        JMP   LABEL1
        .
        .
        .
        JMP   LABEL2
    
```

Both JMP instructions cause program control to be transferred to the same LDD instruction.

The label assigned to an instruction or data definition has as its value the address of the first byte of the instruction or data. Instructions elsewhere in the program can refer to this address by its symbolic label name.

### 2.3.2 Operation or Opcode Field

The operation field is mandatory in every noncomment statement and contains a mnemonic that defines an assembler operation (directive) or machine operation (executable instruction) to be performed.

The operation field may begin in any column and is terminated by a blank, tab, or carriage return, if no operand or comment field is present. Figure 2-5 identifies the operation field in a program.

```

      Operation Field
CLRRAM: CLRA           ;WE MUST DO WHAT WE
        XABR           ;MUST DO
CLR:    CLRA           ;CLEAR ALL RAM
        XIS
        JP            CLR
        XABR
        AISC          13
        JP            CLRRAM
        RC
        XAS           ;TURN OFF SK
        LEI           8 ;ENABLE SHIFT REG
    
```

Figure 2-5. Operation Field in a Source Program

### 2.3.3 Operand Field

The operand field contains additional information (e.g., parameters, immediate data, addresses) required by the assembler to interpret the opcode field completely. The operands may be symbols, constants, or expressions. The operand field must be separated from the operation field by at least one blank. Figure 2-6 identifies the operand fields of a program.

```

      Operand Field
CLRRAM: CLRA           ;WE MUST DO WHAT WE
        XABR           ;MUST DO
CLR:    CLRA           ;CLEAR ALL RAM
        XIS
        JP            CLR
        XABR
        AISC          13
        JP            CLRRAM
        RC
        XAS           ;TURN OFF SK
        LEI           8 ;ENABLE SHIFT REG
    
```

Figure 2-6. Operand Field in a Source Program

### 2.3.4 Comment Field

The comment field is optional and provides additional information that makes the source program easier to read. This field is ignored by the assembler and generates no object code. Comments should be included throughout the program to explain subroutine linkage, assumptions made, algorithms used, formats of inputs, etc.

The following conventions apply to comments:

1. A comment must be preceded by a semicolon(;).
2. All valid characters, including blanks, may be used in comments.
3. Comments should not extend beyond column 80, but a comment may be carried over on the following line (preceded by a semicolon).

Figure 2-7 identifies the comment fields on a program.

```

      Comment Field
CLRRAM: CLRA           ;WE MUST DO WHAT WE
        XABR           ;MUST DO
CLR:    CLRA           ;CLEAR ALL RAM
        XIS
        JP            CLR
        XABR
        AISC          13
        JP            CLRRAM
        RC
        XAS           ;TURN OFF SK
        LEI           8 ;ENABLE SHIFT REG
    
```

Figure 2-7. Comment Field in a Source Program

### 2.3.5 Aligning Fields

One or more spaces are allowed to separate fields. Figure 2-8 illustrates the source program with a single space separating each field of the instruction. For clarity, it is recommended that all fields be aligned at the same character positions in every line.

```

CLRA ; WE MUST DO WHAT WE MUST DO
CLRRAM: XABR ; CLEAR ALL RAM
CLR: CLRA
XIS
JP CLR
XABR
AISC 13
JP CLRRAM
RC
XAS ; TURN OFF SK
LEI 8 ; ENABLE SHIFT REG
    
```

Figure 2-8. Source Program with Unaligned Fields



# Instruction Set

## 3.1 Introduction

This chapter provides information on the instruction sets of the COP400 microcontrollers. As with the architecture of the different devices in the COP400 family, the instruction sets of the various devices allow the user to choose among several devices to provide only as much software capability as is needed for a particular application.

The ASMCOP assembles code for all members of the COP400 Family (410/411/420/421/444/445). Each member of the family is specified by the "CHIP" directive. Instructions being assembled are checked for correct register bounds, address range and legality.

The symbols used in the instruction descriptions are given below:

## 3.2 COP420 Series/COP444L Instruction Set

Table 3-1 provides the mnemonic, operand, machine code, data flow, skip conditions, and description associated with each instruction in the COP420 series/COP444L instruction set. As indicated, an asterisk in the description column signifies a double-byte instruction. Also, notes are provided following this table which describe or refer to additional information relevant to particular instructions. As indicated by Note 3, the ININ and INIL instructions are not included in the COP421 instruction set, due to its lack of IN inputs and the IL3 and IL0 latches associated with two of the IN inputs (IN3 and IN0, respectively).

Note that the COP420 series/COP444L set, as with all COP400 instruction sets, is divided into the following categories: Arithmetic Operations, Input/Output Instructions, Transfer of Control Instructions, Memory Reference Instructions, Register Reference Instructions, and Test Instructions.

Symbol	Definition
a	10- (9- or 11-) bit operand field, 0-1024 binary (ROM address)
d	4-bit operand field, 1-15 binary (RAM digit select)
r	2- (or 3-) bit operand field, 0-3 binary (RAM register select)
RAM(s)	Contents of RAM location addressed by s
ROM(t)	Contents of ROM location addressed by t
y	4-bit operand field, 0-15 binary (immediate data)
A	4-bit accumulator
B	6- (or 7-) bit RAM address register
Bd	Lower four bits of B (digit address)
Br	Upper 2- (or 3-) bits of B (register address)
C	1-bit carry register
D	4-bit data output port
EN	3-bit enable register
G	4-bit register to latch data for G I/O port
IL	Two 1-bit latches associated with the IN (3) or IN (0) inputs
IN	4-bit input port
L	8-bit TRI-STATE® I/O port
M	4-bit contents of RAM memory pointed to by B register
PC	10- (9- or 11-) bit ROM address register (program counter)
Q	8-bit register to latch data for L I/O Port
SA	10- (11- or 9-) bit subroutine save register A
SB	10- (11- or 9-) bit subroutine save register B
SC	10- (or 11-) bit subroutine save register C
SIO	4-bit shift register and counter
SK	Logic-controlled clock output
<b>Optional Symbol</b>	
+	Plus
-	Minus
→	Replaces
=	Is equal to
↔	Is exchanged with
$\bar{A}$	Ones complement of A
⊕	Exclusive OR
:	Range of values

Table 3.1 COP420 Series/COP444L Instruction Set

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
ARITHMETIC INSTRUCTIONS						
ASC		30	$\boxed{0\ 0\ 1\ 1 0\ 0\ 0\ 0}$	$A + C + RAM(B) \rightarrow A$ Carry $\rightarrow C$	Carry	Add with Carry, Skip on Carry
ADD		31	$\boxed{0\ 0\ 1\ 1 0\ 0\ 0\ 1}$	$A + RAM(B) \rightarrow A$	None	Add RAM to A
ADT		4A	$\boxed{0\ 1\ 0\ 0 1\ 0\ 1\ 0}$	$A + 10_{10} \rightarrow A$	None	Add Ten to A
AISC	y	5y	$\boxed{0\ 1\ 0\ 1 y}$	$A + y \rightarrow A$	Carry	Add Immediate, Skip on Carry (y $\neq$ 0)
CASC		10	$\boxed{0\ 0\ 0\ 1 0\ 0\ 0\ 0}$	$\overline{A} + RAM(B) + C \rightarrow A$ Carry $\rightarrow C$	Carry	Complement and Add with Carry, Skip on Carry
CLRA		00	$\boxed{0\ 0\ 0\ 0 0\ 0\ 0\ 0}$	$0 \rightarrow A$	None	Clear A
COMP		40	$\boxed{0\ 1\ 0\ 0 0\ 0\ 0\ 0}$	$\overline{A} \rightarrow A$	None	Ones complement of A to A
NOP		44	$\boxed{0\ 1\ 0\ 0 0\ 1\ 0\ 0}$	None	None	No Operation
RC		32	$\boxed{0\ 0\ 1\ 1 0\ 0\ 1\ 0}$	"0" $\rightarrow C$	None	Reset C
SC		22	$\boxed{0\ 0\ 1\ 0 0\ 0\ 1\ 0}$	"1" $\rightarrow C$	None	Set C
XOR		02	$\boxed{0\ 0\ 0\ 0 0\ 0\ 1\ 0}$	$A \oplus RAM(B) \rightarrow A$	None	Exclusive-OR RAM with A
TRANSFER OF CONTROL INSTRUCTIONS						
JID		FF	$\boxed{1\ 1\ 1\ 1 1\ 1\ 1\ 1}$	ROM (PC <sub>10:8</sub> ,A,M) $\rightarrow$ PC <sub>7:0</sub>	None	Jump Indirect (Note 2)
JMP	a	60-67 00-FF	$\boxed{0\ 1\ 1\ 0 0\ 0 a_{10:8}}$ $\boxed{a_{7:0}}$	$a \rightarrow PC$	None	* Jump
JP	a	00-0E	$\boxed{1\ 1 a_{6:0}}$ (pages 2,3 only)	$a \rightarrow PC_{6:0}$	None	Jump within Page (Note 3)
JP	a	C0-FE	$\boxed{1\ 1 a_{5:0}}$ (all other pages)	$a \rightarrow PC_{5:0}$	None	Jump within Page
JSRP	a	80-8E	$\boxed{1\ 0 a_{5:0}}$	PC+1 $\rightarrow$ SA $\rightarrow$ SB $\rightarrow$ SC 0010 $\rightarrow$ PC <sub>10:6</sub> a $\rightarrow$ PC <sub>5:0</sub>	None	Jump to Subroutine Page (Note 4)
JSR	a	68-6F 00-FF	$\boxed{0\ 1\ 1\ 0 1 a_{10:8}}$ $\boxed{a_{7:0}}$	PC+1 $\rightarrow$ SA $\rightarrow$ SB $\rightarrow$ SC a $\rightarrow$ PC	None	* Jump to Subroutine
RET		48	$\boxed{0\ 1\ 0\ 0 1\ 0\ 0\ 0}$	SC $\rightarrow$ SB $\rightarrow$ SA $\rightarrow$ PC	None	Return from Subroutine
RETSK		49	$\boxed{0\ 1\ 0\ 0 1\ 0\ 0\ 1}$	SC $\rightarrow$ SB $\rightarrow$ SA $\rightarrow$ PC	Always Skip on Return	Return from Subroutine then Skip

Table 3.1 COP420 Series/COP444L Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>MEMORY REFERENCE INSTRUCTIONS</b>						
CAMQ		33 3C	$\begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$	A → Q7:4 RAM(B) → Q3:0	None	• Copy A, RAM to Q
CQMA		33 2C	$\begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$	Q7:4 → RAM(B) Q3:0 → A	None	• Copy Q to RAM, A
LD	r	05,15,25, 35	$\begin{bmatrix} 0 & 0 &   & r &   & 0 & 1 & 0 & 1 \end{bmatrix}$	RAM(B) → A Br ⊕ r → Br	None	Load RAM into A, Exclusive-OR Br with r
LDD	r,d	23 00-7F	$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 &   & r &   & d &   & & \end{bmatrix}$	RAM(r,d) → A	None	• Load A with RAM pointed to directly by r,d
LQID		BF	$\begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$	ROM(PC10:8,A,M) → Q SB → SC	None	Load Q Indirect (Note 3)
RMB	0 1 2 3	4C 45 42 43	$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$	0 → RAM(B) <sub>0</sub> 0 → RAM(B) <sub>1</sub> 0 → RAM(B) <sub>2</sub> 0 → RAM(B) <sub>3</sub>	None	Reset RAM Bit
SMB	0 1 2 3	4D 47 46 48	$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$	1 → RAM(B) <sub>0</sub> 1 → RAM(B) <sub>1</sub> 1 → RAM(B) <sub>2</sub> 1 → RAM(B) <sub>3</sub>	None	Set RAM Bit
STII	y	7y	$\begin{bmatrix} 0 & 1 & 1 & 1 &   & y &   & \end{bmatrix}$	y → RAM(B) Bd + 1 → Bd	None	Store Memory Immediate and Increment Bd
X	r	06,16,26, 36	$\begin{bmatrix} 0 & 0 &   & r &   & 0 & 1 & 1 & 0 \end{bmatrix}$	RAM(B) ↔ A Br ⊕ r → Br	None	Exchange RAM with A, Exclusive-OR Br with r
XAD	r,d	23 80-FF	$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 &   & r &   & d &   & & \end{bmatrix}$	RAM(r,d) ↔ A	None	• Exchange A with RAM pointed to directly by r,d
XDS	r	07,17,27, 37	$\begin{bmatrix} 0 & 0 &   & r &   & 0 & 1 & 1 & 1 \end{bmatrix}$	RAM(B) ↔ A Bd - 1 → Bd Br ⊕ r → Br	Bd decrements past 0	Exchange RAM with A and Decrement Bd, Exclusive-OR Br with r
XIS	r	04,14,24, 34	$\begin{bmatrix} 0 & 0 &   & r &   & 0 & 1 & 0 & 0 \end{bmatrix}$	RAM(B) ↔ A Bd + 1 → Bd Br ⊕ r → Br	Bd increments past 15	Exchange RAM with A and Increment Bd, Exclusive-OR Br with r
<b>REGISTER REFERENCE INSTRUCTIONS</b>						
CAB		50	$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$	A → Bd	None	Copy A to Bd
CBA		4E	$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$	Bd → A	None	Copy Bd to A
LBI	r,d	00 33 80-FF	$\begin{bmatrix} 0 & 0 &   & r &   & (d-1) \\ (d = 0, 9:15) \\ \text{or} \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 &   & r &   & d &   & & \end{bmatrix}$ (any d)	r,d → B	Skip until not a LBI	Load B immediate with r,d (Single-byte)  • Load B Immediate with r,d (Double-byte)
LEI	y	33 6y	$\begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 &   & y &   & \end{bmatrix}$	y → EN	None	• Load EN Immediate (Note 7)
XABR		12	$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$	A ↔ Br (0,0 → A <sub>3</sub> ,A <sub>2</sub> )	None	Exchange A with Br

Table 3.1 COP420 Series/COP444L Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>TEST INSTRUCTIONS</b>						
SKC		20	<u>0 0 1 0</u>   <u>0 0 0 0</u>		C = "1"	Skip if C is True
SKE		21	<u>0 0 1 0</u>   <u>0 0 0 1</u>		A = RAM(B)	Skip if A Equals RAM
SKGZ		33	<u>0 0 1 1</u>   <u>0 0 1 1</u>		G <sub>3:0</sub> = 0	* Skip if G is Zero (all 4 bits)
		21	<u>0 0 1 0</u>   <u>0 0 0 1</u>			
SKGBZ		33	<u>0 0 1 1</u>   <u>0 0 1 1</u>	1st byte	G <sub>0</sub> = 0 G <sub>1</sub> = 0 G <sub>2</sub> = 0 G <sub>3</sub> = 0	* Skip if G Bit is Zero
	0	01	<u>0 0 0 0</u>   <u>0 0 0 1</u>			
	1	11	<u>0 0 0 1</u>   <u>0 0 0 1</u>	2nd byte		
	2	03	<u>0 0 0 0</u>   <u>0 0 1 1</u>			
3	13	<u>0 0 0 1</u>   <u>0 0 1 1</u>				
SKMBZ	0	01	<u>0 0 0 0</u>   <u>0 0 0 1</u>		RAM(B) <sub>0</sub> = 0	Skip if RAM Bit is Zero
	1	11	<u>0 0 0 1</u>   <u>0 0 0 1</u>		RAM(B) <sub>1</sub> = 0	
	2	03	<u>0 0 0 0</u>   <u>0 0 1 1</u>		RAM(B) <sub>2</sub> = 0	
	3	13	<u>0 0 0 1</u>   <u>0 0 1 1</u>		RAM(B) <sub>3</sub> = 0	
SKT		41	<u>0 1 0 0</u>   <u>0 0 0 1</u>		A time-base counter carry has occurred since last test	Skip on Timer (Note 3)
<b>INPUT/OUTPUT INSTRUCTIONS</b>						
ING		33	<u>0 0 1 1</u>   <u>0 0 1 1</u>	G → A	None	* Input G Ports to A
		2A	<u>0 0 1 0</u>   <u>1 0 1 0</u>			
ININ		33	<u>0 0 1 1</u>   <u>0 0 1 1</u>	IN → A	None	* Input IN Inputs to A (Note 2)
		28	<u>0 0 1 0</u>   <u>1 0 0 0</u>			
INIL		33	<u>0 0 1 1</u>   <u>0 0 1 1</u>	IL <sub>3</sub> , "1", "0", IL <sub>0</sub> → A	None	* Input IL Latches to A (Note 3)
		20	<u>0 0 1 1</u>   <u>0 0 1 1</u>			
INL		33	<u>0 0 1 1</u>   <u>0 0 1 1</u>	L <sub>7:4</sub> → RAM(B)	None	* Input L Ports to RAM, A
		2E	<u>0 0 1 0</u>   <u>1 1 1 0</u>	L <sub>3:0</sub> → A		
OBD		33	<u>0 0 1 1</u>   <u>0 0 1 1</u>	Bd → D	None	* Output Bd to D Outputs
		3E	<u>0 0 1 1</u>   <u>1 1 1 0</u>			
OGI	y	33	<u>0 0 1 1</u>   <u>0 0 1 1</u>	y → G	None	* Output to G Ports Immediate
		5y	<u>0 1 0 1</u>   <u>y</u>			
OMG		33	<u>0 0 1 1</u>   <u>0 0 1 1</u>	RAM(B) → G	None	* Output RAM to G Ports
		3A	<u>0 0 1 1</u>   <u>1 0 1 0</u>			
XAS		4F	<u>0 1 0 0</u>   <u>1 1 1 1</u>	A ↔ SIO, C → SK	None	Exchange A with SIO (Note 3)

**Note 1:** All subscripts for alphabetical symbols indicate bit numbers unless explicitly defined (e.g., Br and Bd are explicitly defined). Bits are numbered 0 to N where 0 signifies the least significant (low-order, right-most bit). For example, A<sub>3</sub> indicates the most significant (left-most) bit of the 4-bit A register.

**Note 2:** The ININ instruction is not available on the 24-pin COP421 since this device does not contain the IN inputs.

**Note 3:** For additional information on the operation of the XAS, JID, LQID, INIL, and SKT instructions, see Section 3.2.

**Note 4:** The JP instruction allows a jump, while in subroutine pages 2 or 3, to any ROM location within the two-page boundary of pages 2 or 3. The JP instruction, otherwise, permits a jump to a ROM location within the current 64-word page. JP may not jump to the last word of a page.

**Note 5:** A JSRP transfers program control to subroutine page 2 (0010 is loaded into the upper 4 bits of P). A JSRP may not be used when in pages 2 or 3. JSRP may not jump to the last word in page 2.

**Note 6:** LBI is a single-byte instruction if d = 0, 9, 10, 11, 12, 13, 14, or 15. The machine code for the lower 4 bits equals the binary value of the "d" data minus 1, e.g., to load the lower four bits of B (Bd) with the value 9 (1001<sub>2</sub>), the lower 4 bits of the LBI instruction equal 8 (1000<sub>2</sub>). To load 0, the lower 4 bits of the LBI instruction should equal 15 (1111<sub>2</sub>).

**Note 7:** Machine code for operand field y for LEI instruction should equal the binary value to be latched into EN, where a "1" or "0" in each bit of EN corresponds with the selection or deselection of a particular function associated with each bit. (See Functional Description, EN Register.)

Table 3-2. COP410L/COP411L Instruction Set

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description								
ARITHMETIC INSTRUCTIONS														
ASC		30	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	1	1	0	0	0	0	$A + C + RAM(B) \rightarrow A$ Carry $\rightarrow C$	Carry	Add with Carry, Skip on Carry
0	0	1	1	0	0	0	0							
ADD		31	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	1	1	0	0	0	1	$A + RAM(B) \rightarrow A$	None	Add RAM to A
0	0	1	1	0	0	0	1							
AISC	y	5y	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>1</td><td colspan="4">y</td></tr></table>	0	1	0	1	y				$A + y \rightarrow A$	Carry	Add Immediate, Skip on Carry (y $\neq$ 0)
0	1	0	1	y										
CLRA		00	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	0	0	0	0	0	0	0	$0 \rightarrow A$	None	Clear A
0	0	0	0	0	0	0	0							
COMP		40	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	1	0	0	0	0	0	0	$\bar{A} \rightarrow A$	None	Ones complement of A to A
0	1	0	0	0	0	0	0							
NOP		44	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr></table>	0	1	0	0	0	1	0	0	None	None	No Operation
0	1	0	0	0	1	0	0							
RC		32	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	0	0	1	1	0	0	0	1	"0" $\rightarrow C$	None	Reset C
0	0	1	1	0	0	0	1							
SC		22	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table>	0	0	1	0	0	0	1	0	"1" $\rightarrow C$	None	Set C
0	0	1	0	0	0	1	0							
XOR		02	<table border="1"><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></table>	0	0	0	0	0	0	1	0	$A \oplus RAM(B) \rightarrow A$	None	Exclusive-OR RAM with A
0	0	0	0	0	0	1	0							

TRANSFER OF CONTROL INSTRUCTIONS

JID		FF	<table border="1"><tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr></table>	1	1	1	1	1	1	1	1	ROM (PC <sub>8</sub> ,A,M) $\rightarrow$ PC <sub>7:0</sub>	None	Jump Indirect (Note 2)								
1	1	1	1	1	1	1	1															
JMP	a	60-67 00-FF	<table border="1"><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>a</td></tr></table> <table border="1"><tr><td colspan="8">a<sub>7:0</sub></td></tr></table>	0	1	1	0	0	0	0	a	a <sub>7:0</sub>								a $\rightarrow$ PC	None	* Jump
0	1	1	0	0	0	0	a															
a <sub>7:0</sub>																						
JP	a	80-BE	<table border="1"><tr><td>1</td><td colspan="7">a<sub>6:0</sub></td></tr></table> (pages 2,3 only) or	1	a <sub>6:0</sub>							a $\rightarrow$ PC <sub>6:0</sub>	None	Jump within Page (Note 3)								
1	a <sub>6:0</sub>																					
JP		C0-FE	<table border="1"><tr><td>1</td><td>1</td><td colspan="6">a<sub>5:0</sub></td></tr></table> (all other pages)	1	1	a <sub>5:0</sub>						a $\rightarrow$ PC <sub>5:0</sub>	None	Jump within Page								
1	1	a <sub>5:0</sub>																				
JSRP	a	80-8E	<table border="1"><tr><td>1</td><td>0</td><td colspan="6">a<sub>5:0</sub></td></tr></table>	1	0	a <sub>5:0</sub>						PC+1 $\rightarrow$ SA $\rightarrow$ SB 1010 $\rightarrow$ PC <sub>8:6</sub> a $\rightarrow$ PC <sub>5:0</sub>	None	Jump to Subroutine Page (Note 4)								
1	0	a <sub>5:0</sub>																				
JSR	a	68-69 00-FF	<table border="1"><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>a<sub>8</sub></td></tr></table> <table border="1"><tr><td colspan="8">a<sub>7:0</sub></td></tr></table>	0	1	1	0	1	0	0	a <sub>8</sub>	a <sub>7:0</sub>								PC+1 $\rightarrow$ SA $\rightarrow$ SB a $\rightarrow$ PC	None	* Jump to Subroutine
0	1	1	0	1	0	0	a <sub>8</sub>															
a <sub>7:0</sub>																						
RET		48	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr></table>	0	1	0	0	1	0	0	0	SB $\rightarrow$ SA $\rightarrow$ PC	None	Return from Subroutine								
0	1	0	0	1	0	0	0															
RETSK		49	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr></table>	0	1	0	0	1	0	0	1	SB $\rightarrow$ SA $\rightarrow$ PC	Always Skip on Return	Return from Subroutine then Skip								
0	1	0	0	1	0	0	1															

Table 3-2. COP410L/COP411L Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
MEMORY REFERENCE INSTRUCTIONS						
CAMQ		33 3C	<u>0 0 1 1</u>   <u>0 0 1 1</u> <u>0 0 1 1</u>   <u>1 1 0 0</u>	A → Q7:4 RAM(B) → Q3:0	None	* Copy A, RAM to Q
LD	r	05,15,25, 35	<u>0 0</u>  r  <u>0 1 0 1</u>	RAM(B) → A Br ⊕ r → Br	None	Load RAM into A, Exclusive-OR Br with r
LQID		BF	<u>1 0 1 1</u>   <u>1 1 1 1</u>	ROM(PC8,A,M,) → Q SB → SC	None	Load Q Indirect (Note 3)
RMB	0 1 2 3	4C 45 42 43	<u>0 1 0 0</u>   <u>1 1 0 0</u> <u>0 1 0 0</u>   <u>0 1 0 1</u> <u>0 1 0 0</u>   <u>0 0 1 0</u> <u>0 1 0 0</u>   <u>0 0 1 1</u>	0 → RAM(B) <sub>0</sub> 0 → RAM(B) <sub>1</sub> 0 → RAM(B) <sub>2</sub> 0 → RAM(B) <sub>3</sub>	None	Reset RAM Bit
SiwB	0 1 2 3	4D 47 46 48	<u>0 1 0 0</u>   <u>1 1 0 1</u> <u>0 1 0 0</u>   <u>0 1 1 1</u> <u>0 1 0 0</u>   <u>0 1 1 0</u> <u>0 1 0 0</u>   <u>1 0 0 0</u>	1 → RAM(B) <sub>0</sub> 1 → RAM(B) <sub>1</sub> 1 → RAM(B) <sub>2</sub> 1 → RAM(B) <sub>3</sub>	None	Set RAM Bit
STII	y	7y	<u>0 1 1 1</u>   y	y → RAM(B) Bd + 1 → Bd	None	Store Memory Immediate and Increment Bd
X	r	06,16,26, 36	<u>0 0</u>  r  <u>0 1 1 0</u>	RAM(B) ↔ A Br ⊕ r → Br	None	Exchange RAM with A, Exclusive-OR Br with r
XAD	3,15	23 BF	<u>0 0 1 0</u>   <u>0 0 1 1</u> <u>1 0</u>   <u>1 1</u>   <u>1 1 1 1</u>	RAM(3,15) ↔ A	None	* Exchange A with RAM (3,15)
XDS	r	07,17,27, 37	<u>0 0</u>  r  <u>0 1 1 1</u>	RAM(B) ↔ A Bd - 1 → Bd Br ⊕ r → Br	Bd decrements past 0	Exchange RAM with A and Decrement Bd, Exclusive-OR Br with r
XIS	r	04,14,24, 34	<u>0 0</u>  r  <u>0 1 0 0</u>	RAM(B) ↔ A Bd + 1 → Bd Br ⊕ r → Br	Bd increments past 15	Exchange RAM with A and Increment Bd, Exclusive-OR Br with r
REGISTER REFERENCE INSTRUCTIONS						
CAB		50	<u>0 1 0 1</u>   <u>0 0 0 0</u>	A → Bd	None	Copy A to Bd
CBA		4E	<u>0 1 0 0</u>   <u>1 1 0 0</u>	Bd → A	None	Copy Bd to A
LBI	r,d	00	<u>0 0</u>  r  <u>(d-1)</u> (d = 0,9:15)	r,d → B	Skip until not a LBI	Load B Immediate with r,d (Single-Byte) (Note 5)
LEI	y	33 6y	<u>0 0 1 1</u>   <u>0 0 1 1</u> <u>0 1 1 0</u>   y	y → EN	None	* Load EN Immediate. (Note 6)



Table 3-2. COP410L/COP411L Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
TEST INSTRUCTIONS						
SKC		20	<u>0 0 1 0</u> <u>0 0 0 0</u>		C = "1"	Skip if C is True
SKE		21	<u>0 0 1 0</u> <u>0 0 0 1</u>		A = RAM(B)	Skip if A Equals RAM
SKGZ		33	<u>0 0 1 1</u> <u>0 0 1 1</u>		G <sub>3:0</sub> = 0	* Skip if G is Zero (all 4 bits)
		21	<u>0 0 1 0</u> <u>0 0 0 1</u>			
SKGBZ		33	<u>0 0 1 1</u> <u>0 0 1 1</u>	1st byte	G <sub>0</sub> = 0 G <sub>1</sub> = 0 G <sub>2</sub> = 0 G <sub>3</sub> = 0	* Skip if G Bit is Zero
	0	01	<u>0 0 0 0</u> <u>0 0 0 1</u>	} 2nd byte		
	1	11	<u>0 0 0 1</u> <u>0 0 0 1</u>			
	2	03	<u>0 0 0 0</u> <u>0 0 1 1</u>			
3	13	<u>0 0 0 1</u> <u>0 0 1 1</u>				
SKMBZ	0	01	<u>0 0 0 0</u> <u>0 0 0 1</u>		RAM(B) <sub>0</sub> = 0	Skip if RAM Bit is Zero
	1	11	<u>0 0 0 1</u> <u>0 0 0 1</u>		RAM(B) <sub>1</sub> = 0	
	2	03	<u>0 0 0 0</u> <u>0 0 1 1</u>		RAM(B) <sub>2</sub> = 0	
	3	13	<u>0 0 0 1</u> <u>0 0 1 1</u>		RAM(B) <sub>3</sub> = 0	
INPUT/OUTPUT INSTRUCTIONS						
ING		33	<u>0 0 1 1</u> <u>0 0 1 1</u>	G → A	None	* Input G Ports to A
		2A	<u>0 0 1 0</u> <u>1 0 1 0</u>			
INL		33	<u>0 0 1 1</u> <u>0 0 1 1</u>	L <sub>7:4</sub> → RAM(B)	None	* Input L Ports to RAM, A
		2E	<u>0 0 1 0</u> <u>1 1 1 0</u>	L <sub>3:0</sub> → A		
OBD		33	<u>0 0 1 1</u> <u>0 0 1 1</u>	Bd → D	None	* Output Bd to D Outputs
		3E	<u>0 0 1 1</u> <u>1 1 1 0</u>			
OMG		33	<u>0 0 1 1</u> <u>0 0 1 1</u>	RAM(B) → G	None	* Output RAM to G Ports
		3A	<u>0 0 1 1</u> <u>1 0 1 0</u>			
XAS		4F	<u>0 1 0 0</u> <u>1 1 1 1</u>	A ↔ SIO, C → SK	None	Exchange A with SIO (Note 3)

**Note 1:** All subscripts for alphabetical symbols indicate bit numbers unless explicitly defined (e.g., Br and Bd are explicitly defined). Bits are numbered 0 to N where 0 signifies the least significant (low-order, right-most bit). For example, A<sub>3</sub> indicates the most significant (left-most) bit of the 4-bit A register.

**Note 2:** For additional information on the operation of the XAS, JID, and LQID instructions, see Section 3.2.

**Note 3:** The JP instruction allows a jump, while in subroutine pages 2 or 3, to any ROM location within the two-page boundary of pages 2 or 3. The JP instruction, otherwise, permits a jump to a ROM location within the current 64-word page. JP may not jump to the last word of a page.

**Note 4:** A JSRP transfers program control to subroutine page 2 (0010 is loaded into the upper 4 bits of P). A JSRP may not be used when in pages 2 or 3. JSRP may not jump to the last word in page 2.

**Note 5:** LBI is a single-byte instruction if d=0, 9, 10, 11, 12, 13, 14, or 15. The machine code for the lower 4 bits equals the binary value of the "d" data minus 7, e.g., to load the lower four bits of B (Bd) with the value 9 (1001<sub>2</sub>), the lower 4 bits of the LBI instruction equal 8 (1000<sub>2</sub>). To load 0, the lower 4 bits of the LBI instruction should equal 15 (1111<sub>2</sub>).

**Note 6:** Machine code for operand field y for LEI instruction should equal the binary value to be latched into EN, where a "1" or "0" in each bit of EN corresponds with the selection or deselection of a particular function associated with each bit. (See Functional Description, EN Register.)

### 3.3 COP410L/COP411L Instruction Set

The COP410L and COP411L instruction sets are subsets of the COP421 series instruction set.

Table 3-2 provides the mnemonic, operand, machine code, data flow, skip conditions and description associated with each instruction in the COP410L and COP411L instruction sets. An asterisk in the description column indicates the double-byte instruction. Notes are provided following this table which include additional information relevant to particular instructions.

### 3.4 Arithmetic Instructions

**ASC** Add with Carry, Skip on Carry  
 byte 1  $\boxed{0|0|1|1|0|0|0|0}$  30  
 $A + C + \text{RAM}(B) \rightarrow A$   
 $\text{Carry} \rightarrow C$

ASC (Add with carry, Skip on Carry) performs a binary addition of A, C (carry bit), and M, placing the result in A and C. If a carry occurs, the next program instruction is skipped.

CYCLES: 1  
 SKIP CONDITIONS: Carry

**ADD** Add RAM to A  
 byte 1  $\boxed{0|0|1|1|0|0|0|1}$  31  
 $A + \text{RAM}(B) \rightarrow A$

ADD (ADD) performs binary addition. The 4-bit addends are A and M. The 4-bit sum is placed in A. ADD does not affect the carry or skip.

CYCLES: 1  
 SKIP CONDITIONS: None

**ADT** Add Ten to A  
 byte 1  $\boxed{0|1|0|0|1|0|1|0}$  4A  
 $A + 10_{10} \rightarrow A$

ADT (Add Ten to A) adds ten ( $10_{10}$ ) to A and, like ADD, does not affect the carry or skip. ADT facilitates Binary Coded Decimal (BCD) arithmetic. For example, the following sequence of instructions perform a single-digit BCD add of the contents of A and M (the carry is assumed set when entering this routine if addition of the previous least significant digits produced an overflow ( $A > 9$ )):

ASC 6  
 ASC  
 ADT

CYCLES: 1  
 SKIP CONDITIONS: None

**AISC y** Add Immediate, Skip on Carry ( $y \neq 0$ )  
 byte 1  $\boxed{0|1|0|1| \quad y \quad}$  5y  
 $A + y \rightarrow A$

AISC (Add Immediate, Skip on Carry) adds the instruction operand constant "y" changed. This instruction finds frequent use in BCD add and subtract routines (see ADT and CASC descriptions) as well as in testing the value of A. (If A is greater than 12, for instance, an AISC5 will skip the next instruction.)

This instruction is also used to put a constant in the accumulator.

CYCLES: 1  
 SKIP CONDITIONS: Carry

**CASC** Complement and Add with Carry, Skip on Carry  
 byte 1  $\boxed{0|0|0|1|0|0|0|0}$  10  
 $A + \text{RAM}(B) + C \rightarrow A$

CASC (Complement and Add, Skip on Carry) performs a binary subtraction of A from M by cumming the complement of A (A) with C and M, placing the result in A and C. If no carry out occurs (indicating a borrow), C is reset and the next instruction is executed. If a carry occurs (indicating no borrow) C is set and the next instruction is skipped.

A single BCD digit binary subtraction of A from M may be performed as follows (the carry bit is assumed set upon initial entry to the routine):

CASC  
 ADT

The CASC instruction sets C and skips the ADT instruction if the subtraction does not result in a borrow ( $A > M$ ). If a borrow occurs, the ADT instruction is executed, readjusting the result to the proper BCD value, leaving C reset for propagation of the borrow in the subtraction of the next most-significant BCD digit. CASC is functionally equivalent to a COMP instruction followed by an ASC.

CYCLES: 1  
 SKIP CONDITIONS: Carry

**CLRA** Clear A  
 byte 1  $\boxed{0|0|0|0|0|0|0|0}$  00  
 $0 \rightarrow A$

CLRA (CLear A) clears the accumulator by placing zeros in each of the four bits of A.

This instruction is often required prior to loading A equal to a desired value with an AISC instruction if the previous contents of A are unknown. For instance, to load  $A = 11$ , the following sequence may be necessary:

CLRA  
 AISC 11

The skip features associated with AISC need not be considered in this example (a carry will never occur).

CYCLES: 1  
SKIP CONDITIONS: None

COMP

Ones complement of A to A

byte 1 0|1|0|0|0|0|0|0 40  
A → A

COMP (COMplement A) changes the state of each bit of A with ones becoming zeros and zeros becoming ones. It has the effect of, and may be used to perform, a binary (two's complement) subtraction of A from 15 (1111<sub>2</sub>), e.g., complementing A = 6 (0110<sub>2</sub>) will yield 9 (1001<sub>2</sub>).

CYCLES: 1  
SKIP CONDITIONS: None

NOP

No Operation

byte 1 0|1|0|0|0|1|0|0 44  
None

NOP (No Operation) does not perform any operation. It is useful, however, for simple single instruction time delays or to defeat the skip conditions associated with particular instructions.

CYCLES: 1  
SKIP CONDITIONS: None

RC

Reset C

byte 1 0|0|1|1|0|0|1|0 32  
0 → C

RC (Reset Carry) resets C.

CYCLES: 1  
SKIP CONDITIONS: None

SC

Set C

byte 1 0|0|1|0|0|0|1|0 22  
1 → C

SC (Set Carry) sets C. SC and RC are most often employed to initialize C prior to entering arithmetic routines. They also allow C to be used as a general purpose (testable) flag, as long as subsequent instructions do not inadvertently affect the C register.

CYCLES: 1  
SKIP CONDITIONS: None

XOR

Exclusive-OR RAM with A

byte 1 0|0|0|0|0|0|1|0 02  
A ⊕ RAM(B) → A

XOR (eXclusive-OR A with M) performs a logical Exclusive-OR operation of each bit of A with each corresponding bit of M, placing the result in A. This operation can be used to change the state of any bit in M, if the corresponding (equally weighted) bit of A is set. This follows from the Exclusive-OR truth table where an X + "1" = X, and an X + "0" = X, assuming the "X" bits to be one of the four bits in M, and the "1" and "0" to be equally weighted bits in A. This instruction, therefore, allows the selective complementing or toggling of one or more bits of M.

For example, to change the state of bit 2 of M, set A = 0100, perform an XOR, then exchange A into M with an X instruction.

CYCLES: 1  
SKIP CONDITIONS: None

### 3.5 Transfer of Control Instructions

JID

Jump Indirect

byte 1 1|1|1|1|1|1|1|1 FF  
ROM (PC<sub>10:8</sub>, A, M) →  
PC<sub>7:0</sub>

JID (Jump InDirect) is an indirect addressing instruction, transferring program control to a new ROM location addressed by the contents of the ROM location pointed to by A and M. Specifically, it loads the lower 8-bits of the ROM address register P with the contents of ROM pointed to by the 11-bit word P<sub>10</sub>P<sub>9</sub>P<sub>8</sub>A<sub>3</sub>A<sub>2</sub>A<sub>1</sub>A<sub>0</sub>M<sub>3</sub>M<sub>2</sub>M<sub>1</sub>M<sub>0</sub>. The contents of the selected ROM location (I<sub>7</sub>-I<sub>0</sub>) are, therefore, located into P<sub>7</sub>-P<sub>0</sub>, changing the lower eight bits of P to transfer program control to the new ROM location.

P<sub>10</sub>, P<sub>9</sub> and P<sub>8</sub> remain unchanged throughout the execution of the JID instruction. JID, therefore, may only jump to a ROM location within the current 4-page ROM "block" (page 0-3, 4-7, 8-11, 12-15, 16-19, 20-23, etc.).

JID can be useful in keyboard-decode routines when the values associated with the row and column of a particular key closure are placed in A and M for a jump indirect to the contents of ROM which point to the starting address of the appropriate routine associated with that particular key closure.

CYCLES: 2  
SKIP CONDITIONS: None



**RET** Return from Subroutine  
 byte 1 

0	1	0	0	1	0	0	0
---	---	---	---	---	---	---	---

 48  
 SC→SB→SA→PC

RET (RETURN from subroutine) returns program control to the main program following a JSR or JSRP instruction or interrupt. RET "pops" the stack (SC→SB→SA→P); the next main program instruction address (P + 1) saved in SA is loaded into P, the contents of SB are loaded into SA and the contents of SC are loaded into SB (the contents of SC are also retained in SC). Program control, therefore, is returned to the instruction immediately following the previous subroutine call.

CYCLES: 1  
 SKIP CONDITIONS: None

**RETSK** Return from Subroutine then Skip  
 byte 1 

0	1	0	0	1	0	0	1
---	---	---	---	---	---	---	---

 49  
 SC→SB→SA→PC

RETSK (RETurn from subroutine then SKip), as with the RET instruction above, pops the stack (SC→SB→SA→P), restoring program control to the main program following a subroutine call. However, it always skips the first instruction encountered when it returns to the main program. This instruction provides the programmer with an alternate return from subroutines, either via a RET or RETSK, based upon tests made within the subroutine itself.

CYCLES: 1  
 SKIP CONDITIONS: Always Skip on Return

### 3.6 Memory Reference Instructions

**CAMQ** Copy A, RAM to Q  
 byte 1 

0	0	1	1	0	0	1	1
---	---	---	---	---	---	---	---

 33  
 byte 2 

0	0	1	1	1	1	0	0
---	---	---	---	---	---	---	---

 3C  
 A→Q<sub>7:4</sub>  
 RAM(B)→Q<sub>3:0</sub>

CAMQ (Copy A, M to Q) transfers the 8-bit contents of A and M to the Q latches. A<sub>3</sub>-A<sub>0</sub> are output to Q<sub>7</sub>-Q<sub>4</sub>; M<sub>3</sub>-M<sub>0</sub> are output to Q<sub>3</sub>-Q<sub>0</sub>. Note that CAMQ is the inverse of CQMA (see CQMA instruction) with respect to the r bits of Q with which A and M communicate. Therefore, the input and processing of Q must often be followed by an X (Exchange M with A) instruction, before final output to Q, in order to maintain the proper bit weights of the Q data. For example, the following instructions read Q to M, A, set Q<sub>7</sub> and perform the necessary exchange before execution of the CAMQ instruction:

CQMA ; Q to M, A  
 SMB 3 ; SET Q7 BIT LOCATED IN M3  
 X ; EXCHANGE M WITH A  
 CAMQ ; A, M TO Q

CYCLES: 2  
 SKIP CONDITIONS: None

**CQMA** Copy Q to RAM, A  
 byte 1 

0	0	1	1	1	0	0	1	1
---	---	---	---	---	---	---	---	---

 33  
 byte 2 

0	0	1	0	1	1	0	0
---	---	---	---	---	---	---	---

 2C  
 Q<sub>7:4</sub>→RAM(B)  
 Q<sub>3:0</sub>→A

CQMA (Copy Q to M,A) transfers the 8-bit contents of the Q latches to M and A. Q<sub>7</sub>-Q<sub>4</sub> are placed in M<sub>3</sub>-M<sub>0</sub>; Q<sub>3</sub>-Q<sub>0</sub> are placed in A<sub>3</sub>-A<sub>0</sub>. CQMA can be employed after an LQID (Load Q InDirect) instruction to input or alter the value of lookup data. CQMA is also an essential instruction when the COP420 is employed as a MICROBUST<sup>TM</sup> peripheral component. In such applications, IN3 is used by the control microprocessor to write bus data from the L ports to the Q latches. A CQMA then inputs this data to M,A for processing by the COP420 program.

CYCLES: 2  
 SKIP CONDITIONS: None

**LD r** Load RAM into A  
 byte 1 

0	0	r	0	1	0	1
---	---	---	---	---	---	---

 05, 15  
 RAM(B)→Q 25, 35

LD (LoaD M into A) loads M (the 4-bit contents of RAM pointed to by the B register: M<sub>3</sub>-M<sub>0</sub>) into A<sub>3</sub>-A<sub>0</sub>. After M is loaded into A, the 2-bit "r" operand field is Exclusive-ORed with the contents of Br (upper two bits of B—RAM register select) to point to a new RAM register for successive memory reference operations. Since the properties of the Exclusive-OR logic operation are such that a 1 ⊕ X equals the complement of X, use of the "r" field allows the programmer to switch between any one of the four RAM registers by complementing the appropriate bits of the current contents of the Br register. Of course, if "r" = 0, the contents of Br will remain unchanged after the execution of an LD instruction.

For example, if the assembly language instruction LD 3 ("r" = 11<sub>2</sub>) is executed with Br = 2 (10<sub>2</sub>) and Bd = 12 (1100<sub>2</sub>), the contents of RAM register 2, digit 12 will be loaded to A and Br will be changed to (11<sub>2</sub> + 10<sub>2</sub> = 01<sub>2</sub>), with B pointing to RAM register 1 digit 12. For assembly language programming, use of an Exclusive-OR "r" operand field with memory reference instructions which use this field is optional—if not specified, an "0" operand is assumed.

CYCLES: 1  
 SKIP CONDITIONS: None

**LDD r,d** Load A with RAM  
 byte 1 

0	0	1	0	0	0	1	1
---	---	---	---	---	---	---	---

 23  
 byte 2 

0	r	d					
---	---	---	--	--	--	--	--

 00-7F  
 RAM(r,d) → A for 420/421 (00-3F)

LDD (Load A with M Directly) loads the 4-bit contents of the RAM memory location pointed to directly by the "r" and "d" operand fields (register and digit select, respectively) of the instruction M<sub>3</sub>-M<sub>0</sub>, into A<sub>3</sub>-A<sub>0</sub>. Note that this instruction and the XAD instruction differ from other memory reference instructions in that the operand of the instruction, not the B register, is used to point to the appropriate RAM digit location to be accessed—the B register is unaffected by these instructions. This instruction is useful in accessing RAM counters, status and flag digits, etc., within routines of loops without destroying the previous value of B, allowing the latter to be used for sequential memory access operation and for other reiterative purposes.

CYCLES: 2  
 SKIP CONDITIONS: None

**LQID** Load Q Indirect  
 byte 1 

1	0	1	1	1	1	1	1
---	---	---	---	---	---	---	---

 BF  
 RAM(PC<sub>10:8</sub>,A,M) → Q  
 SB → SC

LQID (Load Q InDirect) is, in effect, a ROM data "lookup" instruction. It translates Q<sub>7</sub>-Q<sub>0</sub>, respectively. It does this by pushing the stack (P + 1 → SA → SB → SC) and replacing the least significant 8 bits of P as follows: A<sub>3</sub>-A<sub>0</sub> → P<sub>7</sub>-P<sub>4</sub>; M<sub>3</sub>-M<sub>0</sub> → P<sub>3</sub>-P<sub>0</sub>, leaving the three most significant bits of P unchanged. The ROM data pointed to by the new P address is fetched and loaded into the Q latches, Q<sub>7</sub>-Q<sub>0</sub>. Next, the stack is popped (SC → SB → SA → P), restoring the previous pushed value of P (P + 1) to continue sequential program execution. Since LQID pushes SB → SC, the previous contents of SC are lost. Also, when LQID pops the stack, the previously pushed contents of SB are left in SC as well as loaded back into SB. The net result, therefore, of an LQID instruction upon the subroutine-save stack is that the contents of SB are placed in SC (SB → SC). Since it pushes the stack, a LQID should not be executed when three levels of subroutine nesting are currently in effect. (The last return address in SC will be lost.)

CYCLES: 2  
 SKIP CONDITIONS: None

**RMB 0** Reset RAM Bit 0  
 byte 1 

0	1	0	0	1	1	0	0
---	---	---	---	---	---	---	---

 4C  
 0 → RAM(B)<sub>0</sub>

**RMB 1** Reset RAM Bit 1  
 byte 1 

0	1	0	0	0	1	0	1
---	---	---	---	---	---	---	---

 45  
 0 → RAM(B)<sub>1</sub>

**RMB 2** Reset RAM Bit 2  
 byte 1 

0	1	0	0	0	0	1	0
---	---	---	---	---	---	---	---

 42  
 0 → RAM(B)<sub>2</sub>

**RMB 3** Reset RAM Bit 3  
 byte 1 

0	1	0	0	0	0	1	1
---	---	---	---	---	---	---	---

 43  
 0 → RAM(B)<sub>3</sub>

RMB (Reset Memory Bit) resets a bit in M as specified by the operand field of the instructions. (Remember, M is the 4-bit RAM digit pointed to by the B register.) The operand field is specified according to the bit number (0-3, left-most to right-most bit) of the particular bit to be reset.

CYCLES: 1  
 SKIP CONDITIONS: None

**SMB 0** Set Ram Bit 0  
 byte 1 

0	1	0	0	0	1	1	0
---	---	---	---	---	---	---	---

 4D  
 1 → RAM(B)<sub>0</sub>

**SMB 1** Set RAM Bit 1  
 byte 1 

0	1	0	0	0	0	1	1
---	---	---	---	---	---	---	---

 47  
 1 → RAM(B)<sub>1</sub>

**SMB 2** Set RAM Bit 2  
 byte 1 

0	1	0	0	0	0	1	1
---	---	---	---	---	---	---	---

 46  
 1 → RAM(B)<sub>2</sub>

**SMB 3** Set RAM Bit 3  
 byte 1 

0	1	0	0	0	1	0	0
---	---	---	---	---	---	---	---

 48  
 1 → RAM(B)<sub>3</sub>

SMB (Set Memory Bit) sets a bit in M as specified by the operand field of the instructions. (Remember, M is the 4-bit RAM digit pointed to by the B register.) The operand field is specified according to the bit number (0-3, left-most to right-most bit) of the particular bit to be set, e.g., an SMB 3 would set the most significant bit of M. These instructions are useful in operating upon program status flags located in RAM.

CYCLES: 1  
 SKIP CONDITIONS: None

**STII y** Store Memory Immediate and Increment B  
 byte 1 

0	1	1	1				
---	---	---	---	--	--	--	--

 y 7y  
 y → RAM(B)  
 Bd + 1 → Bd

STII (Store Memory Immediate and Increment Bd) loads the r-bit contents specified by the "y" operand field of the instruction into the RAM memory digit pointed to by the B register, M3-M0. It is important to note that the value of Bd (RAM digit-select) is incremented (as with the XIS instruction) after the "y" data is stored in M.

CYCLES: 1  
SKIP CONDITIONS: None

X r

Exchange RAM with A

byte 1 

0	0	r	0	1	1	1	0
---	---	---	---	---	---	---	---

 06, 16,  
RAM(B) ↔ A 26, 36  
Br or → Br

X (eXchange M with A) exchanges the 4-bit contents of RAM pointed to by the B register, M3-M0, with A3-A0. The "r" operand field of the instruction is Exclusive-ORed with the contents of BR after the exchange to provide a new Br RAM register select value as explained in the LD instruction previously.

CYCLES: 1  
SKIP CONDITIONS: None

XAD rd

Exchange RAM with A and Decrement Bd

byte 1 

0	0	1	0	0	0	1	1
---	---	---	---	---	---	---	---

 23  
byte 2 

1	r	d
---	---	---

 80-FF  
RAM(r,d) ↔ A

XAD (eXchange A with M Directly) exchanges the 4-bit contents of the RAM memory location pointed to directly by the "r" and "d" operand fields of the instruction, M3-M0, with A3-A0. It has the same characteristics and utility as the LDD instruction, e.g., the B register is not affected.

CYCLES: 2  
SKIP CONDITIONS: None

XDS r

Exchange RAM with A and Decrement Bd, Exclusive-OR Br with r

byte 1 

0	0	r	0	1	1	1
---	---	---	---	---	---	---

 07, 17,  
RAM(B) ↔ A 27, 37  
Bd - 1 → Bd  
Br or → Br

XDS (eXchange M with A, Decrement Bd and Skip on borrow) performs the same operation as the X instruction above, and also decrements the value of the Bd register (RAM digit-select) after the exchange. Use of an "r" operand field will result in both an altered RAM digit-select value and a new RAM register select value in B. XDS skips the next program instruction when Bd is decremented past 0 (after the contents of RAM digit 0 have been exchanged with A and XDS decrements Bd

to 15). Repeated XDSs will "walk down" through the digits of a RAM register before skipping. XDS together with X instructions can be used to operate upon the corresponding digits of different RAM registers in successive fashion.

CYCLES: 1  
SKIP CONDITIONS: Bd decrements past 0

XIS r

Exchange RAM with A and Increment Bd

byte 1 

0	0	r	0	1	0	0
---	---	---	---	---	---	---

 04, 14,  
RAM(B) ↔ A 24, 34  
Bd + 1 → Bd  
Br or → Br

XIS (eXchange M with A, Increment Bd, and Skip on Carry) performs the same operation as the XDS instruction except that it increments Bd after the exchange and skips the next program instruction after Bd increments past 15 (after the contents of RAM digit 15 have been exchanged with A and XIS increments Bd to 0). Consequently, successive XISs "walk up" through the digits of a RAM register before skipping.

CYCLES: 1  
SKIP CONDITIONS: Bd increments past 15

### 3.7 Register Reference Instructions

CAB

Copy A to Bd

byte 1 

0	1	0	1	0	0	0	0
---	---	---	---	---	---	---	---

 50  
A → Bd

CAB (Copy A to Bd) transfers the 4-bit contents of A, A3-A0, to Bd (the RAM digit-select register). This instruction allows the loading of a new RAM digit-select value via the accumulator, a useful operation in many memory-digit access loops.

CYCLES: 1  
SKIP CONDITIONS: None

CBA

Copy Bd to A

byte 1 

0	1	0	0	1	1	1	0
---	---	---	---	---	---	---	---

 4E  
Bd → A

CBA (Copy Bd to A) transfers the 4-bit contents of Bd (RAM digit-select) to A3-A0. It is the functional complement of the CAB instruction and finds similar use in memory-digit access loops.

CYCLES: 1  
SKIP CONDITIONS: None

LBI r,d

Load B Immediate (single-byte)

byte 1 

0	0	r	(d-1)
---	---	---	-------

 00  
or (d = 0,9:15)

## Load B Immediate (double-byte)

byte 1 

0	0	1	1	1	0	0	1	1
---	---	---	---	---	---	---	---	---

 33 (any d)  
 byte 2 

1	r	d
---	---	---

 80-FF  
 r,d → B

LBI (Load B Immediate) loads the B register with the 7-bit value specified by the "r" (2-bit) and "d" (4-bit) fields of the instruction. Its purpose is to directly load a new RAM register and digit select value into B and, unlike CAB, CBA or XABR, does not require use of the accumulator. A further distinction with respect to CAB and CBA is its ability to alter the Br register (RAM register-select).

The LBI instruction is coded or assembled into machine language as either a single- or a double-byte instruction, depending on the value of the "d" field. If the "d" field value equals 0 or 9 through 15, the instruction is coded as a single-byte instruction with the lower six bits equal to the value of "d" minus 1. If the "d" field equals 1 through 8 (1-8), the instruction is coded as a double-byte instruction, with the lower six bits of the second byte equal to the value of "d".

To take advantage of the more efficient single-byte LBI format, frequently used program data (counters, flags, etc.) should be placed within RAM digit locations accessible by the LBI single-byte "d" field (d = 0, 9-15).

An important characteristic of the LBI instruction is that it will skip all subsequent LBI instructions until it encounters an instruction which is not an LBI. This feature accommodates it for use in multiple-entry subroutines.

CYCLES: 1 or 2

SKIP CONDITIONS: Skip until not an LBI

LEI y

## Load EN Immediate

byte 1 

0	0	1	1	1	0	0	1	1
---	---	---	---	---	---	---	---	---

 33  
 byte 2 

0	1	1	0	y
---	---	---	---	---

 6y  
 y → EN

LEI (Load EN Immediate) loads the enable register with the value contained in the "y" operand field of this instruction (0-15, binary). Its function is to select or deselect a particular software selectable feature associated with each of the four bits of the enable register (EN3-EN0). These features and the corresponding bit weights and values associated with each feature are as follows:

1. The least significant bit of the enable register, EN0, selects the SIO register as either a 4-bit shift register or a 4-bit binary counter. With EN0 set, SIO is an asynchronous binary counter, decrementing its value by one upon each low-going pulse ("1" to "0") occurring on the SI input. Each pulse must remain at each logic level at least two instruction cycles. SK outputs the value of the C upon the execution of an XAS and remains latched until the execution of another XAS instruction. The SO output is equal to the value of EN3.

With EN0 reset, SIO is a serial shift register, shifting continuously left each instruction cycle time. The data present at SI goes into the least significant bit of SIO; SO can be enabled to output the most significant bit of SIO each cycle time. SK output becomes a logic-controlled clock, providing a SYNC signal each instruction time. It will start outputting a SYNC signal each instruction time. It will start outputting a SYNC pulse upon the execution of an XAS instruction with C = "1", stopping upon the execution of a subsequent XAS with C = "0".

If EN0 is changed from "1" to "0" ("0" to "1"), the SK output will change from "1" to SYNC (SYNC to "1") without the execution of an XAS instruction.

2. With EN1 set, the IN1 input is enabled as an interrupt input upon the occurrence of a negative pulse on IN1; program control is transferred to the last word of page 3 (address OFF<sub>16</sub>). Immediately following an interrupt, EN1 is reset to disable further interrupts until later set by an LEI instruction (usually at the end of the interrupt service routine or later within the main program).

The following features are associated with the IN1 interrupt procedure and protocol and must be considered by the programmer when using this software selectable feature of the COP420 series. (Interrupt is unavailable on the COP421 series since it does not have the IN3-IN0 inputs).

- a. The interrupt, once acknowledged as explained below, pushes the next sequential program counter address (P + 1) onto the stack, pushing in turn the contents of the other subroutine-save registers to the next lower level (P + 1 → SA → SB → SC). Any previous contents of SC are lost. The program counter is set to address OFF<sub>16</sub> (the last word of page 3) and EN1 is reset.
- b. An interrupt will be acknowledged only after the following conditions are met:
  - 1) EN1 has been set.
  - 2) A low-going pulse ("1" to "0") at least two instruction cycles in width has occurred on the IN1 input.
  - 3) A currently executing instruction has been completed.



- 4) All successive transfer of control instructions and successive LBI's have been completed (e.g., if the main program is executing a JP instruction which transfers program control to another JP instruction, the interrupt will not be acknowledged until the second JP instruction has been executed).
- c. Upon acknowledgement of an interrupt, the skip logic status is saved and implemented upon popping the stack during the execution of a subsequent RET instruction. For example, if an interrupt occurs during the execution of ASC (Add with carry, Skip on Carry) instruction which results in a carry, the next instruction (which would normally be skipped) is not skipped; instead, its address is pushed onto the stack; the skip logic status is saved and program control is transferred to the interrupt servicing routine at location OFF<sub>16</sub>. At the end of the interrupt routine, a RET instruction is executed to pop the stack and return program control to the instruction following the original ACS. At this time, the skip logic is enabled and skips this instruction because of the previous ACS carry. Since, as explained above, it is the RET instruction which enables the previously saved status of the skip logic, subroutines should not be nested within the interrupt service routine since their RET instruction will enable any previously saved main program skips, interfering with the orderly execution of the interrupt routine. Also, the LQID instruction should not be used within any interrupt routine because it pops the stack and thus will enable any previously saved main program skips.
- d. The first instruction of the interrupt routine at address OFF<sub>16</sub> must be NOP.
3. With EN2 set, the L drivers are enabled, loading data previously latched into Q to the L I/O ports. Resetting EN2 disables the L drivers, placing the L I/O ports in a high-impedance state. When the L I/O ports are used as segment drivers to an LED display, the setting and resetting of EN2 results in the outputting and blanking, respectively, of segment data to the display. When using the MICROBUS™ option, EN2 does not affect the L drivers.
4. EN3, in conjunction with EN0, affects the SO output. With EN0 set (binary counter option selected) SO will output the value loaded into EN3. With EN0 reset (serial shift register feature selected), setting EN3 enables SO as the output of the SIO shift register, outputting serial shifted data (the most significant bit of SIO) each instruction time as explained above. Resetting EN3 with the serial shift register feature selected disables SO as the shift register output; data continues to be shifted through SIO and can be exchanged with A via an XAS instruction, but SO remains reset to "0".

CYCLES: 2  
SKIP CONDITIONS: None

**XABR** Exchange A with Br  
byte 1 0 0 0 0 1 1 0 0 0 1 0 12  
A ↔ Br(0→A<sub>3</sub>)

XABR (eXchange A with Br) exchanges Br (upper three bits of B: RAM register-select) with A. Since Br contains only 3 bits, only the lower 3 bits of A, A<sub>2</sub>-A<sub>0</sub>, are placed in Br. Similarly, the 3 bits of Br are placed in A<sub>2</sub>-A<sub>0</sub> with a zero being loaded into the upper bit of A, A<sub>3</sub>. XABR is an efficient means of loading the Br register via the accumulator; a direct load of the Br register must otherwise be accomplished by an LBI instruction which also affects the Bd portion of the B register.

CYCLES: 1  
SKIP CONDITIONS: None

### 3.8 Test Instructions

**SKC** Skip If C is True  
byte 1 0 0 0 1 1 0 0 0 0 0 0 20

SKC (SKip on Carry) skips the next program instruction if the carry bit is equal to one. When used in conjunction with the RC and SC instructions, it allows C to be used as a 1-bit testable flag.

CYCLES: 1  
SKIP CONDITIONS: C = "1"

**SKE** Skip If A Equals RAM  
byte 1 0 0 0 1 1 0 0 0 0 0 1 21

SKE (SKip if A Equals M) compares all four bits of A with M, skipping the next instruction if the value of A is equal to the value of M. SKE can be used to compare A with a status or counter digit in M, skipping to an instruction which transfers program control to another routine if equality exists.

CYCLES: 1  
SKIP CONDITIONS: A = RAM(B)

**SKGZ** Skip If G is Zero  
byte 1 0 0 1 1 1 0 0 1 1 1 33  
byte 2 0 0 1 1 0 0 0 0 0 1 21

SKGZ (SKip If G is Zero) is a double-byte instruction. It tests the state of all four of the G lines, skipping the next program instruction if G<sub>3</sub>-G<sub>0</sub> are equal to zero.

CYCLES: 2  
SKIP CONDITIONS: G<sub>3:0</sub> = 0

**SKGBZ** Skip If G Bit is Zero  
byte 1 0 0 1 1 1 0 0 1 1 1 33

**SKGBZ 0** byte 2 0 0 0 0 0 0 0 0 0 1 01

**SKGBZ 1** byte 2 0 0 0 0 1 1 0 0 0 1 11

SKGBZ 2 byte2 0|0|0|0|0|0|1|1 03SKGBZ 3 byte2 0|0|0|1|0|0|1|1 13

SKGBZ (SKip if G Bit is Zero) is a double-byte instruction. It tests the state of one of the four G lines ( $G_3$ - $G_0$ ) as specified by the "n" operand of the instruction, skipping the next program instruction if the specified G line is equal to zero.

CYCLES: 2

SKIP CONDITIONS:  $G_0 = 0$  $G_1 = 0$  $G_2 = 0$  $G_3 = 0$ 

SKMBZ Skip if RAM Bit is Zero

SKMBZ 0 byte1 0|0|0|0|0|0|0|1 01SKMBZ 1 byte1 0|0|0|1|0|0|0|1 11SKMBZ 2 byte1 0|0|0|0|0|0|1|1 03SKMBZ 3 byte1 0|0|0|1|0|0|1|1 13

SKMBZ (SKip on Memory Bit Zero) skips the next program instruction if the RAM memory bit specified by the "n" field of instruction (0-3, right-most to left-most M bit) is equal to zero. This instruction, together with the SMB and RMB instructions, allow for the testing and manipulation of single-bit flags contained within RAM digit locations.

CYCLES: 1

SKIP CONDITIONS:  $RAM(B)_0 = 0$  $RAM(B)_1 = 0$  $RAM(B)_2 = 0$  $RAM(B)_3 = 0$ 

SKT

Skip on Timer

byte1 0|1|0|0|0|0|0|1 41

SKT (SKip on Timer) instruction tests the state of an internal 10-bit time-base counter. This counter divides the instruction cycle clock frequency by 1024 and provides a latched indication of counter overflow. The SKT instruction tests this latch, executing the next program instruction if the latch is not set. If the latch has been set since the previous test, the next program instruction is skipped and the latch is reset. The features associated with this instruction, therefore, allow the controller to generate its own time-base for real-time processing rather than relying on an external input signal.

For example, using a 2.097 MHz crystal as the time-base to the clock generator, the instruction cycle clock frequency will be 131 kHz (crystal frequency divided by 16) and the binary counter output pulse frequency will be 128 Hz. For time-of-day or similar real-time processing, the SKT instruction can call a routine which increments a "seconds" counter every 128 ticks.

CYCLES: 1

SKIP CONDITIONS: A time-based counter carry has occurred since last test.

# Directives

## 4.1 Introduction

Assembler directives are source statements that the assembler recognizes as directions to perform some particular operation, such as, put a title on each page of the program listing, define data, etc. Directives normally do not generate any executable object code.

The label field and the comment field in a directive are defined exactly like the label and comment fields in an instruction.

The directives are written in the same format as the COPS instructions, and normally can be interspersed throughout your assembly language program.

The following documentation conventions are used in describing the directives:

1. User-supplied directive labels/names and parameters are shown enclosed in angle brackets "[ ]". Items shown enclosed in the brackets "[ ]" are optional.
2. Actual directives are shown in uppercase.
3. The symbol "|" specifies that the item on either side may be used.
4. Items (except for the symbol "|") shown outside the angle brackets are part of the directive syntax.
5. Three consecutive dots "... " indicates optional multiple occurrences of the preceding item.

## 4.2 Directive Format

The directives are coded using the following syntax:

```
[<label>: | <symbol> | <directive mnemonic>
[<parameter>, ... ][:<comments>]
```

where:

*label* is a user-supplied name, terminated by a colon (:) for a directive. It is optional for all directives except for the SET, MACRO, and "=".

*symbol* without the terminating colon (:) is required and used only for SET, MACRO, and "=" directives.

*directive mnemonic* is the mnemonic used for a directive. Only one directive per line can be entered.

*parameter* may be an expression or a character string. Most directives require at least one parameter.

*comments* Optional program documenting comments. When included, they must be preceded by a semicolon ";".

The directives are listed below and explained in detail in the following pages.

Directive	Function
.ADDR	Address constant generation
.BYTE	Define byte
.CHIP	Identification of COP400 device
.CREF	Start cross reference
.DO	Begin DO loop
.ELSE	Conditional assembly directive
.END	Physical end of source program
.ENDDO/.ENDM	End DO loop
.ENDDO/.ENDM	End macro definition
.ENDIF	Conditional assembly directive

## Directive

Directive	Function
.ERROR	Generate error message
.EXIT	Exit DO loop or macro expansion
=	Assignment
.FORM	Output listing top-of-form
.IF	Conditional assembly directive
.IFC	If character directive
.INCLD	Include disk file source code
.LIST	Listing output control
.LOCAL	Begin local region
.MACRO	Begin macro definition
.MLOC	Macro local symbol designation
.OPT	Define COP400 device options
.PAGE	Set location counter to page address
.PRINTX	Send message to CRT screen
.SET	Assign values to variables
.SPACE	Space n lines on output listing
.TITLE	Identification of program
.WORD	8-bit data generation
.=	Change program counter
.XCREF	Stop cross reference

## 4.3 Definition Directives

### 4.3.1 Define 8-Bit Word (.BYTE and .WORD) Directives

Label	Operation	Operand	Comment
[Label:]	.BYTE	expression [,expression ...]	[:comments]
[Label:]	.WORD	expression [,expression ...]	[:comments]

These directives tell the assembler to allocate memory space and assign the value specified by the data in the operand field of this directive. Beginning at the current value of the location counter, data is stored consecutively in memory, one 8-bit byte of data for each given expression.

If the directive has a label, it refers to the address of the first expression.

The value of each expression must be in the range -128 to +127 for signed data or 0 to 255 for unsigned data. Each expression is evaluated to an 8-bit unsigned integer. Each character string must be coded enclosed in single quotes. Expressions and/or strings must be separated from each other by commas. Any combination of expressions and strings may be specified. The directive statement must be contained in one source line. (A source line may be up to 131 characters long.)

The operands comprising the expressions must be defined before the directive is encountered.

Both directives are equivalent to the STARPLEX™ directive DB.

### 4.3.2 Define Address Constant (.ADDR) Directive

Label	Operation	Operand	Comment
[Label:]	.ADDR	expression [,expression ...]	[:comments]

The .ADDR directive generates 8-bit bytes as specified by one or more expressions in the operand field of this directive and places them in successive memory locations. These expressions are usually labels and are used as address pointers by the COP400 JID (Jump-Indirect) instruction which transfers program control to the contents of the address generated by the .ADDR directive. This directive masks out the upper eight bits of the expression specified in the operand field, and the lower eight bits in successive memory locations. Next, the lower eight bits of the symbol or expression are masked and a comparison is made of the upper eight bits with the current location counter address to ensure that the address generated by the .ADDR directive is in the same 4-page ROM block as the assembler location counter. This test is necessary since the JID instruction must access a pointer and transfer program control within the current 4-page ROM "block." If this test indicates an out-of-range expression, an error message is generated upon assembly and listed on the assembler output listing.

#### 4.4 Symbol Definition Directive

##### 4.4.1 Symbol Assignment (.SET) Directive

Label	Operation	Operand	Comment
[Label:]	.SET	symbol,expression	[;comments]

The .SET directive assigns the value of the expression to the symbol. A symbol assigned a value with a .SET directive can be assigned different values an arbitrary number of times within an assembly language program, with each new value taking precedence over the previous value for a particular symbol. The name is encountered in the assembly, and the value of the expression will be used.

This directive is identical to the EQU directive, except that the name may be defined more than once.

This directive is equivalent to the STARPLEX™ directive SET.

Example: .SET A,100 ;Set A = 100  
 .SET C,A-25\*B/4 ;Set C = A-25\*B/24

##### 4.4.2 Assignment Statement

Label	Operation	Operand	Comment
Symbol	=	expression [,expression ...]	[;comments]

The assignment statement assigns the value of the expression on the right of the equals sign to the symbol on the left of the equals sign. If two expressions are given, the value of the left most is shifted by four bits, and the right-most expression, which must be evaluated to less than 16, is added to this value. The assignment statement may also refer to the current value of the location counter. The location symbol (.) may appear on both sides of the assignment statement equals sign.

Example:

```
. = X'20 ;Set location counter to address
;X'20 (hex value 20)
. = . + 10 ;Reserve 10 locations for later use
LOC: . ;Save current location counter value
= . ;in "LOC"
```

The statement

```
. = expression
```

is identical to the STARPLEX ORG directive.

#### 4.5 Assembler Control Directives

##### 4.5.1 Include File (.INCLD) Directive

Label	Operation	Operand	Comment
[Label:]	.INCLD	[:]filename	[;comments]

The .INCLD directive includes the symbolic file specified in the operand field of the directive in the current assembler source code. Specifically, it causes the assembler to read source code from the specified file on the current diskette until an end-of-file mark is reached, at which time it will again start reading source code from the assembly input file.

The colon in front of the filename is required if that filename is not on the diskette in drive 0.

This directive is identical to STARPLEX directive INCLD except that parentheses are not required. Included files may not be nested and may not contain ".END" directive.

##### 4.5.2 Change Location Counter (. =) Directive

Label	Operation	Operand	Comment
	.	= expression	[;comments]

The value of the location counter (defined by the "." symbol) is set to the value of the expression on the right of the "=" sign.

This directive is a special case of the assignment statement. For additional information, see Section 4.4.2.

Example:

```
. = X'100 ;Set location counter to address
;X'100 (hex value 100)
. = . + 20 ;Reserve 20 memory locations
```

This directive is identical to the STARPLEX ORG directive.

##### 4.5.3 Page Address (.PAGE) Directive

Label	Operation	Operand	Comment
[Label:]	.PAGE	[expression]	[;comments]

The .PAGE directive changes the assembler's location counter to the address of the beginning of the ROM page specified by the expression in the operand field. The value of the expression field may not exceed the maximum ROM page number of the chip being used. Default is advancing to the next page.

#### 4.5.4 End of Source (.END) Directive

Label	Operation	Operand	Comment
[Label:]	.End		[comments]

The .END directive signifies the physical end of the source program. All assembly source statements appearing after this directive are ignored. All assembler programs must terminate with the .END directive. This directive is identical to the STARPLEX™ directive END.

#### 4.5.5 Define Local Region (.LOCAL) Directive

Label	Operation	Operand	Comment
[Label:]	.LOCAL		[comments]

The .LOCAL directive establishes a new program section for local labels. All local labels within a local region are defined only within that particular section of the program. Up to 58 .LOCAL directives may appear in one assembly, giving a maximum of 59 local regions. For example, if a program does not contain a .LOCAL directive, then any local symbol is accessible throughout the program. If a program contains one .LOCAL directive, then the program is divided into two local regions, one before the .LOCAL and the other after it.

#### 4.5.6 Title (.TITLE) Directive

Label	Operation	Operand	Comment
[Label:]	.TITLE	symbol, ['string']	[comments]

The .TITLE directive identifies the load module and output listing in which it appears with a symbolic name and an optional definitive title string. If a .TITLE directive does not appear in the program, the load module and output listing are given the name MAINPR. If more than one .TITLE directive is used, the last one encountered specifies the symbolic name.

"string" is a string of up to 80 ASCII characters terminated by a carriage return. The string is printed with the page header on all pages following the specifica-

tion of the title until a new title is specified. The absence of this directive in a program forces a default title to be used in the page header on each page. When a .TITLE directive is encountered, it forces the string following it to appear at the top of all succeeding pages until a new .TITLE directive is encountered.

This directive accomplishes the functions of both the STARPLEX directives TITLE, and NAME.

#### 4.5.7 Top-of-Form (.FORM) Directive

Label	Operation	Operand	Comment
[Label:]	.FORM	['string']	[comments]

The .FORM directive spaces forward to the top of the next page of the output listing (form feed). The optional string is printed as a page subtitle on each page until a .FORM directive containing a new string is encountered. This directive accomplishes the functions of both STARPLEX directives PAGE and SUBTTL.

#### 4.5.8 Space Forward (.SPACE) Directive

Label	Operation	Operand	Comment
[Label:]	.SPACE	expression	[comments]

The .SPACE directive skips forward a number of lines on the output listing as specified by the expression in the operand field.

#### 4.5.9 List (.LIST) Directive

Label	Operation	Operand	Comment
[Label:]	.LIST	expression	[comments]

The .LIST directive controls listing of the source program. Control of the various list options depends upon the state of the six least significant bits of the evaluated expression in the operand field. Options are usually combined to give the desired type of listing. The following table shows the options available, their associated bit weights and assembler default values.

List Options

Control Function	Positions	Binary Value	6-Bit Hex Value	Descriptions
Master List	0	0	00	Suppress all listing
		1	01	Full Listing (default)
.IF List	1	0	00	Suppress listing of unassembled code (default)
		1	02	Full listing of .IFs and .IFCs
Macro List	2,3	00	00	List only macro calls (default)
		10	08	List only code generated by macro calls
		11	0C	List all code expanded during macro calls
Binary List	4	0	00	List only the first two bytes of generated data
		1	10	List all the binary output by statements generating more than one word (default)
Include List	5	0	00	List only error lines for the included file (default)
		1	20	List the included file

#### 4.5.10 Send Message to CRT Screen (.PRINTX) Directive

Label	Operation	Operand	Comment
[Label:]	.PRINTX	dstringd	[comments]

The .PRINTX directive sends "string" to the CRT screen during pass 2. The delimiter "d" may be any nonblank character. This directive is not available in the PDS COPST<sup>TM</sup> assembler, but is available in the STARPLEX<sup>TM</sup> COPS assembler.

#### 4.5.11 Generate Error (.ERROR) Directive

Label	Operation	Operand	Comment
[Label:]	.ERROR	{'string'}	[comments]

The .ERROR directive generates an error message and an assembly error that is included in the error count at the end of the program.

In the PDS COPS assembler, this directive is only valid in macros. However, in the STARPLEX COPS assembler, this directive is valid any time.

#### 4.5.12 Start Cross Reference (.CREF) Directive

Label	Operation	Operand	Comment
[Label:]	.CREF		[comments]

The .CREF directive causes the gathering of the cross reference information to be initiated as if it had been terminated previously by an .XREF directive. This directive has no effect unless .XREF was specified in the command line. .CREF is not available in the PDS COPS assembler, but is available in the STARPLEX COPS assembler.

#### 4.5.13 Stop Cross Reference (.XREF) Directive

Label	Operation	Operand	Comment
[Label:]	.XREF		[comments]

The .XREF directive causes the gathering of the cross reference information to be terminated until a subsequent .XREF directive is encountered. It does not stop the incrementing of the line number. This directive is not available in the PDS COPS assembler, but is available in the STARPLEX COPS assembler.

### 4.6 Repetition Directives

#### 4.6.1 Do Loop (.DO) Directive

Label	Operation	Operand	Comment
[Label:]	.DO	expression	[comments]

The .DO directive indicates the starting of a repetition block. All the text from .DO until corresponding .ENDDO or .ENDM will be repeated "expression" times.

#### 4.6.2 End Do (.ENDDO or .ENDM) Loop

Label	Operation	Operand	Comment
[Label:]	.ENDDO		[comments]
[Label:]	.ENDM		[comments]

This directive is required to terminate a do-loop (repetition block) or a macro. Each .ENDDO or .ENDM terminates the most recent do-loop or macro that has not already been terminated. .ENDDO and .ENDM are identical.

#### 4.6.3 Exit Do (.EXIT) Loop

Label	Operation	Operand	Comment
[Label:]	.EXIT		[comments]

Early termination of looping in a do-loop (repetition block) can be affected with the .EXIT directive. When .EXIT directive is encountered during assembly, the assembler stops expansion and proceeds to the statement immediately following the .ENDDO or .ENDM directive.

If this .ENDM marks the end of a macro definition, assembly resumes at the statement following the macro call. If the .ENDM marks the end of a repetition block, .EXITM terminates not only the current expansion, but subsequent iterations as well. .EXITM can only appear within a macro definition or a repetition block.

### 4.7 Conditional Assembly Directives

The conditional assembly directives allow selective assembly of source code segments depending on whether a specified condition is true or false. The true/false tests are performed by the assembler.

A conditional assembly block begins with an .IF or .IFC directive and terminates with an .ENDIF directive. A conditional assembly block may be divided into two segments by including the .ELSE directive to end the first segment and begin the second segment.

The assembler evaluates the condition specified in the .IF or .IFC directive and then, depending on the result, assembles the code within the conditional assembly block.

#### 4.7.1 If (.IF) Directive

Label	Operation	Operand	Comment
[Label:]	.IF	expression	[comments]

Condition is true if expression evaluates to greater or less than zero, and false if equal to zero.

#### 4.7.2 If Character (.IFC) Directive

Label	Operation	Operand	Comment
[Label:]	.IFC	string1 operator string2	[comments]

The .IFC directive allows conditional assembly based on character strings rather than the value of an expression as in the .IF directive. String1 and String2 are the character strings to be compared. Operator is the relational operator between the strings. Two operators are allowed: EQ (equal) and NE (not equal). If the relational operator is satisfied, the condition is true.

### 4.7.3 Else (.ELSE) Directive

Label	Operation	Operand	Comment
[Label:]	.ELSE		[comments]

The .ELSE directive indicates the code between the .ELSE and the .ENDIF is to be assembled if the .IF condition is false. It corresponds to the most recent .IF directive which has not been terminated by an .ENDIF.

### 4.7.4 End of If (.ENDIF) Directive

Label	Operation	Operand	Comment
[Label:]	.ENDIF		[comments]

The .ENDIF directive terminates an .IF block. It terminates the most recently opened block which is not terminated by a previous .ENDIF.

Any data appearing in the operand field of an .ELSE or .ENDIF directive causes an error.

If the expression is TRUE, all the instructions between the .IF directive and the next .ELSE or .ENDIF directive are assembled. If the expression is FALSE, the instructions are not assembled.

.ELSE is the converse of .IF. If the expression is FALSE, all instructions between .ELSE and the next .ENDIF directive are assembled. If the expression is TRUE, the instructions are not assembled. The .ELSE directive is optional.

All statements between an .IF directive and its associated .ENDIF directive are defined as an .IF-.ENDIF block. .IF-.ENDIF blocks can be nested to eight levels. Only one .ELSE directive can be included in an .IF-.ENDIF block.

#### Example 1:

```
.IF REG EQ 0
    .
    ;ASSEMBLE IF REG = 0 IS TRUE
    .
.ENDIF
```

#### Example 2:

```
.IF REG EQ 0
    .
    ;ASSEMBLE IF REG = 0 IS TRUE
    .ELSE
    .
    ;ASSEMBLE IF REG = 0 IS FALSE
    .
.ENDIF
```

#### Example 3:

```
.IF REG EQ 0
    .
    ;ASSEMBLE IF REG = 0 IS TRUE
    .IF TYP EQ 0
    .
    ;ASSEMBLE IF REG = 0 IS TRUE
    ;AND TYPE = 0 IS TRUE
    .ELSE
    .
    ;ASSEMBLE IF REG = 0 IS TRUE
    ;AND TYP = 0 IS FALSE
    .ENDIF
    .ELSE
    .
    ;ASSEMBLE IF REG = 0 IS FALSE
    .ENDIF
    .ELSE
    .IF TYP EQ 1
    .
    ;ASSEMBLE IF REG = 0 IS FALSE
    ;AND TYPE = 1 IS TRUE
    .ELSE
    .
    ;ASSEMBLE IF REG = 0 IS FALSE
    ;AND TYP = 0 IS FALSE
    .ENDIF
    .ENDIF
```

## 4.8 COPS Instruction Set Directives

### 4.8.1 Define COP400 Device Options (.OPT) Directive

Label	Operation	Operand	Comment
[Label:]	.OPT	expression1, expression2	[comments]

The .OPT directive specifies to the assembler which mask-programmable options have been selected for the device for which the program is written. The first expression indicates the option number; the second expression indicates the value to be assigned to the specified option number. Value of the first expression must be within the range 1 through 52; value for the second expression must be within range 0 through 14.

This directive is not available in the Standard STARPLEX directive. The options available differ according to the chip chosen.

### 4.8.2 Identification of the COP400 Device (.CHIP) Directive

Label	Operation	Operand	Comment
[Label:]	.CHIP	expression	[comments]

The .CHIP directive specifies to the assembler the particular COP device for which the assembly source code is being written. This is necessary since different COP400 devices having a different number of COP400 instructions may use the COP Cross-Assembler. The device which may be specified with the .CHIP directive and the corresponding values for their operand field expressions are shown on the following page.

<b>COP400 Devices</b>	<b>Operand Expression</b>
COP410L	410
COP411L	411
COP420/420L/420C	420
COP421/421L/421C	421
COP444L	444
COP445L	445
COP440/2440	440, 2440
COP441/2441	441, 2441
COP442/2442	442, 2442
COP422	422

If there is no .CHIP directive, then 420 is assumed and a warning message is generated to indicate that assumption. More than one .CHIP directive may be used to switch among instruction sets.



# Macros

## 5.1 Introduction

Programming in simple assembly language enables a user to be as efficient with his microprocessor resources as his capabilities allow. With assembly language, the user can specify explicitly every detail of the program operation. Because of this, a program in assembly language often takes longer to write than the same program written in a high-level language that fills in many details automatically according to its internal design. This design may or may not be compatible with either the language of the machine on which the high-level language operates or the user's problem. Ideally, the user would like a programming language that is compatible with the machine as need be, while remaining as natural as possible for the expression of his particular problem. The language should fill in details whenever they are routine and should leave the user free to specify the details whenever they are crucial. This ideal can often be closely approximated by the user of a versatile programming tool known as macros.

Macros are a form of text replacement that provide an automatic code-generation completely under the user's control. With macros, a user can gradually build a library tailored to his application, and, with a library of macros oriented toward a particular application, a user who is not a software expert can produce efficient machine-language code; and an experienced user can significantly reduce his program development time.

## 5.2 Macro Directives

Table 5-1 contains a list of macro directives. A discussion of each macro follows the table.

Table 5-1. Macro Directives

Name	Mnemonic	Function
Begin macro definition	.MACRO	Defines a macro
End macro definition	.ENDDO	Ends a macro repeat or a macro definition
End macro definition	.ENDM	Ends a macro definition or a macro repeat
Local macro symbols	.MLOC	Defines local macro symbols
Exit macro	.EXIT	Terminates an expansion

### 5.2.1 Begin Macro Definition (.MACRO) Directive

Label	Operation	Operand	Comment
mname	.MACRO	[parameters]	[comments]

Macro is the directive mnemonic which initiates the macro definition. It must be terminated by at least one blank.

"mname" is the name of the macro. It is legal to define a macro with the same name as an already existing macro, in which case the latest definition is operative. The macro name is used by the main program to call the macro, and must adhere to the rules given for symbol construction.

"parameters" is the optional list of formal parameters used in the macro definition. Each parameter must be a valid symbol and successive parameters must be separated by commas or by commas and blanks.

The macro body consists of assembly language statements. The macro body may consist of simple text, text with formal parameters, and/or macro-time operators. At the time of a macro call, each formal parameter is substituted with the value of the corresponding actual parameter.

### 5.2.3 End Macro Definition (.ENDDO and .ENDM) Directives

Label	Operation	Operand	Comment
[Label:]	.ENDDO		[comments]
[Label:]	.ENDM		[comments]

The .ENDM or .ENDDO directive terminates a macro definition. Each macro definition requires a matching .ENDM or .ENDDO. Each .ENDM or .ENDDO terminates the most recent macro or repetition block that has not already been terminated. These directives are identical.

### 5.2.3 Define Local Symbol (.MLOC) Directive

Label	Operation	Operand	Comment
[Label:]	.MLOC	symbol [,symbol...]	[comments]

When a label is defined within a macro, a duplicate definition results with the second and each subsequent call of the macro. This problem can be avoided by using the .MLOC directive to declare labels local to the macro definition. The .MLOC directive may occur at any point in a macro definition, but it must precede the first occurrence of the symbol(s) it declares local.

### 5.2.4 Exit Macro (.EXIT) Directive

Label	Operation	Operand	Comment
[Label:]	.EXIT		[comments]

The .EXIT directive terminates the expansion of a macro or repetition block. When the .EXIT statement is encountered during assembly, the assembler stops and proceeds immediately to the next .ENDM or .ENDDO directive. If the .ENDM or .ENDDO directive marks the end of a macro definition, assembly resumes at the statement following the macro call.

## 5.3 Basic Macro Concepts

The main use of macros is to insert assembly language statements into a source program, as shown in Figure 5-1. In the example, the original source program contains a macro instruction, or macro call, named NONAME. NONAME is a macro that inserts four NOP instructions into the program listing. When the assembler processes NONAME, it inserts the predefined sequence of assembly language from the macro definition named NONAME into the source program immediately after the point of call (NONAME).

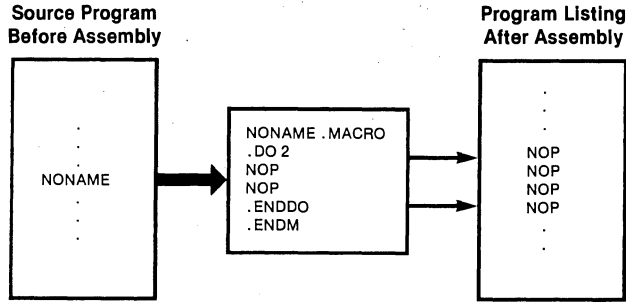


Figure 5-1. Statement Insertion

The process of inserting the text of the macro definition into the source program is called macro expansion. The expanded macro is then processed as if it were part of the original source program. You will note that the macro call itself does not produce any machine language code. The directives used to define the limits to the macro definition are `.MACRO` and `.ENDM`.

Figure 5-1 illustrates three aspects of a macro: the definition, the reference, and the expansion.

### 5.4 Macros and Subroutines

A macro is similar to a subroutine in that it is written once and called many times. A particular programming task may be accomplished by a macro or a subroutine. One technique may be more convenient than the other, depending on the task.

Calling a macro inserts the macro code in the program. Suppose a macro has  $n$  instructions. Calling the macro ten times will result in  $10 \times n$  lines of macro code in the assembled program. By contrast, repeatedly calling a subroutine does not multiply the amount of subroutine code in that assembled program. Sometimes it is necessary to conserve available memory space, and, in that case, the programmer would favor subroutines rather than macros.

The advantage that macros have is that because the program does not have to jump to the subroutine and then return, they will execute faster. So, the choice is usually execution time versus memory space required.

### 5.5 Defining a Macro

Defining a macro involves preparing statements that perform the following functions:

1. Give it a name.
2. Declare any parameters to be used.
3. Write the statements it contains.
4. Establish its boundaries.

The following form is used to define a macro:

```

mname .MACRO [parameters] [;comments]
.
.
macro body
.
.
.ENDM
  
```

where:

- a. "mname" is the name of the macro (the name used to "call" the macro). It is legal to define a macro with the same name as an already existing macro. The latest definition is operative. The macro name must adhere to all rules for symbols.
- b. `.MACRO` is the directive that initiates the macro definition. `.MACRO` must be followed by a blank. Macros must be defined before their use.
- c. Parameters is the operational list of parameters used in the macro definition. The list of parameters must adhere to all the rules for expressions. Parameters are separated by commas or commas and blanks.

The following are examples of legal and illegal `.MACRO` directives.

Legal	Illegal	Reason Illegal
MAC .MACRO A,B	SUB .MACRO \$1\$	Special character is used in parameter.
\$ADD .MACRO OP1,OP2	1MAC .MACRO C,D	First character in macro name is illegal.
LIST .MACRO \$1	MACB .MACRO 25	First character in parameter must be alphabetic or \$.
MSG3 .MACRO	\$AC .MACRO	Special character is used in macro name.

- d. Macro body consists of assembly language statements. The macro body may contain simple text, text with formal parameters and /or macro-time operators. At the time of a macro call, each formal parameter is substituted with the value of the corresponding actual parameter.
- e. The `.ENDM` or `.ENDDO` directive terminates the macro definition.

## 5.6 Calling a Macro

Once a macro has been defined, it then may be called. A macro is called by placing the macro name in the operation field of an assembly language statement, and the parameters in the operand field. The following form is used for a macro call:

```
mname [parameters]
```

where:

- a. "mname" is the name previously assigned in the macro definition.
- b. "parameters" is the list of input parameters. When a macro is defined without parameters, the parameter list is omitted from the call.

## 5.7 Using Parameters

The power of a macro can be increased tremendously through the use of the optional parameters. The parameters allow variable values to be declared when the macro is called. The variable values are then replaced with constants when the MACRO is called.

### 5.7.1 Macro Definition

Macros can be made more powerful through the use of parameters.

Parameters need not be variables or numeric values, but can be any string. The following macro, for example, takes an ASCII string as input and generates a message string in memory suitable for input to the MMSG routine.

```
MSGSTR .MACRO LABEL,STRING
.LABEL: .BYTE 'STRING'
        .BYTE 0
        .ENDM
```

The following macro generates a call to the MMSG routine with the name of the message as input.

```
MESG .MACRO MSGNAM
      JSR 337
      .DBYTE MSGNAM
      .ENDM
```

Note the principle here: the hexadecimal firmware address is maintained centrally in the MESG macro, not scattered all over the code as the macro calls will be.

### 5.7.2 Calling a Macro with Parameters

When parameters are included in a macro call, the following rules apply to the parameter list:

1. Commas or commas and blanks delimit parameters.
2. Consecutive blanks are treated as a single delimiter.
3. A comma leading, following, or imbedded in a string of blanks is treated as a single delimiter.
4. Parameters can be used in the macro definition to define a formal parameter list. The macro statement can use the parameter names in the definition.
5. Parameters can also refer to a parameter by its position in the actual parameter list. During macro expansion, "#n" is replaced by the nth parameter in the list.
6. Parameters may be symbols, numbers, or literal strings.

7. Missing or null parameters are permitted and are treated as strings of zero length.
8. Missing parameters may be omitted at the end of a parameter list.

### 5.7.3 Parameters Referenced by Number

**#**—*Number of Parameters*: '#' is a macro operator that references the parameter list in the macro call. When used in an expression, it is replaced by the number of parameters in the macro call. The following .IF directive, for example, causes the conditional code to be expanded if there are more than the parameters in the macro call:

```
.IF # EQ 10
```

**#N**—*Nth Parameter*: When used with a constant or variable, the '#N' operator references individual parameters in the parameter list. The following example demonstrates how this function is used:

```
X .MACRO
  .BYTE #1,#2,#3
  .ENDM
```

The instruction "X 3,5,2" generates ".BYTE 3,5,2". This relieves the need for naming each parameter in a long list and allows powerful macros to be defined using arbitrary numbers of parameters.

**'**—*Concatenation*: The ' macro operator is used for concatenation. When found, the ' is removed from the output string and the strings on each side of the operator are compressed together after parameter substitution. The following example illustrates use of ' operator.

Macro definition:

```
IMAGINARY .MACRO X
          R X: .BYTE 0
          I X: .BYTE 0
          .ENDM
```

Macro call:

```
IMAGINARY 5
```

Macro expansion:

```
RS: .BYTE 0
IS: .BYTE 0
```

## 5.8 Local Symbols

When a label is defined with a macro, a duplicate definition results with the second and each subsequent call. The problem can be avoided by using the .MLOC directive to declare labels local to the macro definition.

Local symbols are replaced with unique names at expansion time with ZZxxxx, where xxxx is a 4-digit hexadecimal number. The user should avoid using his own labels of the above form as it may cause duplicate definition errors. The .MLOC directive may occur at any point in a macro definition, but it must precede the first occurrence of the symbols it declares local. If

it does not, no error will be reported, but symbols used before the .MLOC will not be recognized as local.

## 5.9 Conditional Expansion

The versatility and the power of the macro assembler is enhanced by the conditional assembly directives. The conditional assembly directives (.IF, .ELSE and .ENDIF) allow the user to generate different lines of code from the same macro simply by varying the parameter values used in the macro calls. Four relational operators are provided:

```
EQ equal
NE not equal
< less than
> greater than
```

### 5.9.1 .IF, .ELSE, .ENDIF Directives

When the macro assembler encounters an .IF directive within a macro expansion, it evaluates the relational operation that follows. If the expression is satisfied (evaluated greater than 0), the lines following the .IF are expanded until an .ELSE or an .ENDIF directive is encountered. If the expression is not satisfied (evaluated less than or equal to 0), only the lines from the .ELSE to the .ENDIF are expanded. See Chapter 4 for additional information on the conditional assembly directives.

### 5.9.2 .IFC Directive

The .IFC directive allows conditional assembly based on character strings rather than the value of an expression as in the .IF directive. String1 and String2 are the character strings to be compared. Operator is the relational operator between the strings. Two operators are allowed: EQ (equal) and NE (not equal). If the relational operator is satisfied, the lines following the .IFC are assembled until an .ELSE or an .ENDIF is encountered. The .ELSE and .ENDIF directives have the same effect with the .IFC directive as they do with the .IF directive.

## 5.10 Macro-Time Looping

Macro-time looping is facilitated through the .DO and .ENDDO directives. These directives are used to delimit a block of statements which are repeatedly assembled. The number of times the block will be assembled is specified on the .DO directive. Following is the format of a .DO-.ENDDO block:

```
.DO count
.
.
.
source
.
.
.ENDDO
```

Note: .DO, .ENDDO, and .EXIT are defined only with a macro definition.

The following examples show the use of the .DO, .ENDDO, and .EXIT directives. The macro CTAB generates a constant table from 0 to MAX where MAX is a parameter of the macro call. Each word has a label DX:—where X is the value of the data word.

```
CTAB      .MACRO      MAX
          .SET        X,0
          .DO         MAX+1
DOX:      .BYTE      X
          .SET        X,X+1
          .ENDDO
          .ENDM
```

Now a call of the form:

```
CTAB      10
```

generates code equivalent to:

```
.SET      X,0
D00:      .BYTE      X
          .SET      X,X+1
D01:      .BYTE      X
          SFT      X,X+1
D02:      .BYTE      X
          .
          .
          .SET      X,X+1
D09:      .BYTE      X
          .SET      X,X+1
D0A:      .BYTE      X
```

### Note

Care must be taken when writing macros that generate a variable number of data words through the use of the .IF or the .DO directives. If the operands on these directives are forward references, their values change between pass 1 and pass 2 and the number of generated words may change. Should this be the case, all labels defined after the macro call that has changed values generate numerous assembly errors of the following form:

```
ERROR DUP DEF
```

## 5.11 Nested Macro Calls

Nested macro calls are allowed. That is, a macro definition may contain a call to another macro. When a macro call is encountered during macro expansion, the state of the macro currently being expanded is saved and expansion begins on the nested macro. Upon completing expansion of the nested macro, expansion of the original macro continues. Depth of nesting allowed will depend on the parameter list sizes, but usually about eight levels of nesting are allowed.

A logical extension of a nested macro call is a recursive macro call; that is, a macro that calls itself. This is allowed, but care must be taken that an infinite loop is not generated.

## 5.12 Nested Macro Definitions

A macro definition can be nested within another macro. Such a macro is not defined until the outer macro is expanded and the nested .MACRO statement is executed. This allows the creation of special purpose macros based on the outer macro's parameters.

# Operating Instructions

## 6.1 Introduction

This chapter describes the COPSTM Cross-Assembler operation. Refer to the STARPLEX™ System Software Reference Manual (Publication No. 420305788) or STARPLEX II™ Software Reference Manual (Publication No. 420306383), chapters 4 and 6 for Text Editor and Linker operation.

Execution of a COPS Assembly Language program involves the following steps:

1. Code a source program.
2. Transcribe the source program to a source file on a diskette using the STARPLEX Text Editor.
3. Assemble the source program to create a load module.

## 6.2 Invoking the Assembler

The assemblers can be invoked from the Command Interpreter using one of two methods:

1. Entering an ASM Command.
2. Using the ASM key (STARPLEX only).

### 6.2.1 Entering an ASM Command

The first method of invoking the assembler is by entering the assembler name followed by the appropriate parameters entered as a single command string. The format of the immediate command line is as follows:

```
ASMCOP source [object [listing [XREF] [ISE]]]
```

Note: If ASMCOP is entered without any parameters, the form is displayed.

### 6.2.2 Using the ASM Key (STARPLEX only)

The second method of invoking the assembler is by pressing the ASM key on the keyboard. This will cause a form like the following one to be displayed on the screen. The user will enter the assembler name as the first parameter, followed by the other appropriate parameters for each field, and then press the RETURN key. The format of the display is as follows:

## 6.2.3 Parameters

The first four parameters of the ASM command are position dependent. The other parameters, XREF and ISE may be entered in any order after the first four. Parameters are separated by one or more spaces. The line is terminated by a carriage return. The input parameters for the assembler are as follows:

### 6.2.3.1 Assembler

The user should enter an ASMCOP as the assembler name. If no extension is specified, none is assumed. When using the ASM key, if a filename is not specified, the default is ASM80.

### 6.2.3.2 Source

"Dev." is the device that contains the source file. If this is not specified, FDSO: is used. "Filename" is the one to six character alphanumeric name of the source file. This is required. The extension ".Ext" is the one to three character identifier that further describes the file function. If this is not specified, ".MAC" is assumed.

### 6.2.3.3 Object

The syntax of the filename is the same as the source; however, if no extension is specified, REL is used. If no filename is specified, the filename specified in the source entry is used, with the extension REL. If no object file is desired, the parameter NIL: is entered to suppress generation.

### 6.2.3.4 Listing

The syntax of the filename is the same as the source; however, if no extension is specified, "LST" is assumed. If listing is to be directed to the printer, LPTn: is entered. If no filename is specified, no listing is generated. If other options are desired, then listing may be suppressed by entering \$NIL or NIL: after the object filename.

<b>STARPLEX Assembler</b>			
Enter data in the appropriate fields and then depress "RETURN"			
Entry Item	Entry	Entry Format	Default
ASSEMBLER	_____	[<dev>:]<filename>[<.ext>]	ASM80
SOURCE	_____	[<dev>:]<filename>[.<ext>]	FDSO: <source> .MAC
OBJECT	_____	[<dev>:]<filename> <ext>]	<Source> .REL
LISTING	_____	[<dev>:]<filename>[<.ext>]	No listing created
XREF	_____	XREF	No cross reference data created
ISE	_____	ISE	No ISE symbol table created

### 6.2.3.5 XREF

If XREF is specified, a list file is created, and the cross reference information is appended to the end of the source listing. It contains an alphabetical list of all user symbols, with a list of all line numbers in which each symbol was referenced. If XREF is not specified, then no cross reference listing is produced.

### 6.2.3.6 ISE™ Symbol Table

If ISE is specified, an ISE symbol table is generated on the same device as the source file. It has the same name as the source file, but with .SYM extension. Default is to generate no ISE symbol table.

Note: 6.2.3.6 does not apply for use with the SPM-A15 product at this time.

## 6.3 Object File Format

The object file module is absolute and can be directly loaded by COP monitor. The requirement for the object field is that it contains the following:

1. Code generated.
2. Chip number.
3. Options from the .OPT directive.
4. Source file checksum.
5. Object checksum.

### 6.3.1 Object File Load Module

The Load Module (LM) file contains loading information and object code produced from the source statements. The LM file is an unformatted file composed of a sequence of records, each containing up to 36 bytes. The representation of the records depends on the storage medium. There are three types of LM records:

- Title record (one per LM file)
- Data record (variable number per LM file)
- End record (one per LM file)

The records are produced in the sequence shown in Figure 6-1. Independent of the record type, the first two bytes in each record always have the same interpretation. The first byte specifies the record type (bits 7 and 6) and the length of the record body (bits 5 through 0). The second byte contains a checksum for error detection. The checksum is formed by taking the arithmetic sum of all the bytes in the record body.

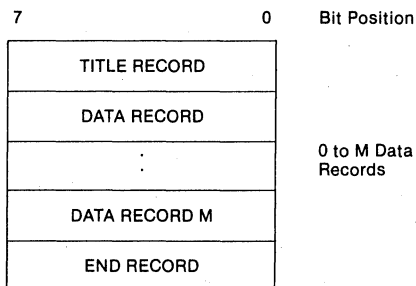


Figure 6-1. LM File Format

### 6.3.2 Title Record

The title record identifies the load module by name and, optionally, by a descriptive character string. These two items are supplied by the last .TITLE directive statement in the source program. If the .TITLE directive is not included, a default name will be the source filename. If the default program name is assigned, the descriptive string is empty. Figure 6-2 illustrates the format of the title record.

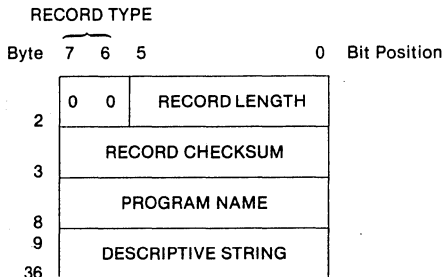


Figure 6-2. Title Record Format

Notes:

1. The program name and descriptive string are composed of 7-bit ASCII characters. The strings are right justified with a zero-fill at the end.
2. Only the first 28 characters in the descriptive string (of the source statement) are used in the title record.

### 6.3.3 Data Record

The data records contain the actual data and instruction bytes to be loaded into memory. Each data record contains the load address of the initial data byte of the record. Each time a discontinuity (empty area or change-of-page) occurs in a program, the current record is terminated and outputted, and a new record is initiated. Figure 6-3 illustrates the format of the data record.

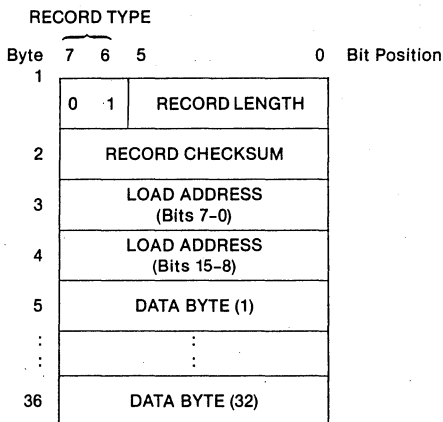


Figure 6-3. Data Record Format

### 6.3.4 End Record

The end record marks the end of the LM file and specifies an entry address for the load module. The format of the end record is illustrated in Figure 6-4. The source checksum represents the sum of all the characters, taken one at a time, in a program source file. The sum is printed on the program listing following the symbol table printout. The object sum represents the sum of all the individual record checksums of the LM. This sum is also printed on the program listing following the symbol table. Also, nonspecified options will be set to "F" Hex.

### 6.4 Listing Format

The time and date appear at the top of every page. Each line of source listing consists of:

- Line number
- Location
- Object code generated
- Source text

Assembly errors are spelled out as messages. (See Appendix D for a complete list of error messages.)

The summary at the end of the listing includes the following:

- Alphabetical listing of all macros defined
- Alphabetical listing of all user symbols and their respective values
- Error count
- Number of ROM words used
- Chip number
- Checksum of source
- Checksum of object
- Filename of source
- Filename of listing

If the user specified XREF, then cross reference information will be appended to the end of the listing.

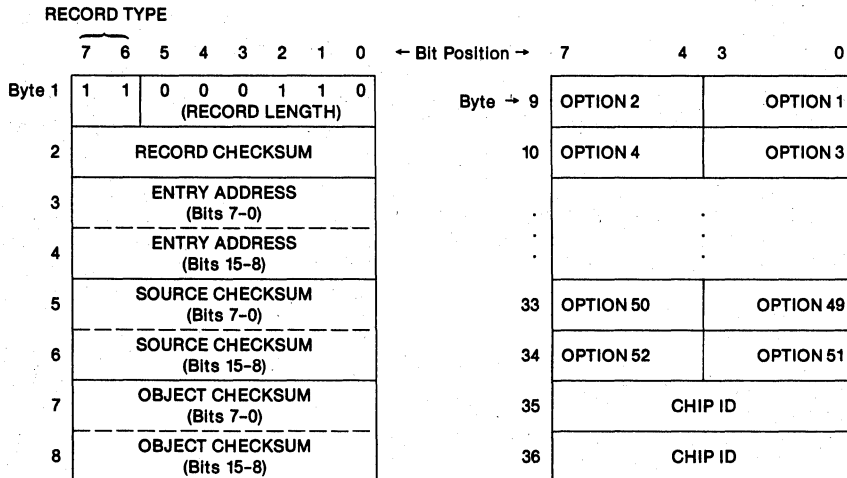


Figure 6-4. End Record Format

### 6.5 COPS™ Cross-Assembler Messages

If a source program assembles error-free, the system displays the following message:

No Fatal Error(s)

If the cross-assembler detects any errors or warnings, the source lines containing errors or warnings are displayed on the screen. Then a message in the following format is displayed:

nn Fatal Error(s) nn Warning(s)

where:

nn is the number of errors detected and/or number of warnings issued by the cross-assembler.

An appropriate message will be printed on the line. Therefore, for each line of source with an error, two lines will be generated in the listing.

A program assembled with fatal errors will not execute. A program assembled with warning(s) may or may not execute; the results may be unpredictable.

If errors are detected in a source program, the source program should be appropriately corrected and the program reassembled.

# Programming Techniques

## 7.1 Introduction

This chapter provides several examples of programming techniques for COP400 devices. All examples are given in COPS™ Cross-Assembler language, using COP400 assembler instruction mnemonics and operand and statements. Although, in the following examples, instruction operands and ROM page numbers are written using decimal notation, the programmer may specify these expressions in hexadecimal notation; the assembler accepts either format (e.g., AISC 13 = AISC X'C, Page X'A = Page 10). On occasion, source code examples contain noninstruction statements, such as assembler directives which convey information to the assembler necessary for proper address allocation or similar assembler-related tasks.

## 7.2 Program Memory Allocation

Generally, COP420 series program memory may be thought of as one area of 1024 bytes of ROM with an address range of 0 to 3FF (hexadecimal). However, while this concept is convenient in writing, in assembling and debugging major portions of COP420 series programs, it is necessary, with respect to a few instructions, to conceptualize program memory on a 64-word "page" basis. Specifically, because of the characteristics and restrictions associated with the JP, JSRP, JID, and LQID instructions, the organization of program memory is as follows:

Chips	Bytes/words	No. of Pages
410/411	512	8, (0-7)
420/421	1024	16, (0-15)
444/445	2048	32, (0-31)

The following discussion provides information and examples relating to the "page" characteristics of each of these unique instructions. Table 7.1 provides a conversion chart indicating the hexadecimal address equivalents for each of the 16 "pages" of ROM. Note: each page consists of 0 through 3F<sub>16</sub> words.

### 7.2.1 JP Instruction

The JP instruction transfers program control to a ROM location whether within a page or within a 2-page boundary consisting of "subroutine pages" 2 or 3.

The following page restrictions apply to the JP instruction.

- When used in any page other than page 2 or 3, it can only jump to a word within the current page.
- When used in page 2 or 3, it may jump to a word within page 2 or 3.
- In all cases, it cannot jump to the last word of a page (word 03F<sub>16</sub>).

The JP instruction assembly operand normally consists of a program label or expression specifying the address of the word to be jumped to. To specify page boundaries and ensure correct placement of the JP and other page-oriented instructions, the assembler .PAGE directive is used to specify the beginning of new page boundaries for program code placement.

The following are examples of use of the JP instruction when used outside subroutine pages 2 and 3.

```

.PAGE 0                ;PLACE FOLLOWING CODE
                       ;IN PAGE 0
LABEL1:
JP LABEL2             ;LEGAL JUMP WITHIN PAGE
.
.
.
LABEL2:
JP LABEL3             ;ILLEGAL JUMP TO LAST
                       ;WORD OF PAGE
JP LABEL4             ;ILLEGAL JUMP TO
                       ;ANOTHER PAGE
.
.
.
LABEL3:
PAGE 1                ;THIS INSTRUCTION IN LAST
                       ;WORD OF PAGE 0
                       ;PLACE FOLLOWING CODE
                       ;ON PAGE 1*
.
.
.
LABEL4:
.
.
.

```

\* Note: The .PAGE 1 directive is not necessary — the COPS Assembler automatically places code in successive memory locations. After a particular page is full, code is automatically placed in successive location on the following page.

The following examples illustrate use of the JP instruction when in subroutine pages 2 and 3:

```

.PAGE 2                ;START OF "SUBROUTINE"
                       ;PAGE 2 CODE
LABEL1:
JP LABEL3             ;LEGAL JUMP TO PAGE 3
                       ;LOCATION
JP LABEL2             ;ILLEGAL JUMP TO LAST
                       ;WORD OF PAGE
.
.
.
LABEL2:
.PAGE 3                ;LAST WORD OF PAGE 2
                       ;START OF PAGE 3 CODE
.
.
.
JP LABEL4             ;ILLEGAL JUMP TO PAGE
                       ;OUTSIDE PAGE 2 OR 3
.
.
.
LABEL3:
JP LABEL1             ;LEGAL JUMP TO PAGE 2
                       ;LOCATION
JP LABEL3             ;LEGAL JUMP WITHIN PAGE
.
.
.
PAGE 4                ;START OF PAGE 4 CODE
.
.
.
LABEL4:
.
.
.
JP LABEL1             ;ILLEGAL JUMP TO PAGE 2
                       ;(MAY ONLY BE DONE WHEN
                       ;IN PAGE 2 OR 3)
.
.
.

```





Table 7-1. Page to Hexadecimal Address Table

Page	Hexadecimal Address Range	Page	Hexadecimal Address Range	Page	Hexadecimal Address Range	Page	Hexadecimal Address Range
0	000-03F	8	200-23F	16	400-43F	24	600-63F
1	040-07F	9	240-27F	17	440-47F	25	640-67F
2	080-08F	10	280-28F	18	480-48F	26	680-68F
3	0C0-0FF	11	2C0-2FF	19	4C0-4FF	27	6C0-6FF
4	100-13F	12	300-33F	20	500-53F	28	700-73F
5	140-17F	13	340-37F	21	540-57F	29	740-77F
6	180-18F	14	380-38F	22	580-58F	30	780-78F
7	1C0-1FF	15	3C0-3FF	23	5C0-5FF	31	7C0-7FF

**7.2.2 JSRP Instruction**

The JSRP instruction is another page-oriented instruction that transfers program control to a word located within "subroutine" page 2 only. Its primary purpose is to allow a single-byte jump to a subroutine in page 2 from any program location other than from page 2 or 3. The JSRP pushes the subroutine-save stack to allow a return to the next program instruction following the subroutine call. The restrictions with the JSRP instructions are as follows:

1. JSRP cannot be used to jump to a subroutine when in pages 2 or 3. (The double-byte JSR instruction can be used for this purpose.)
2. JSRP cannot be used to jump to a subroutine located at the last word of page 2. (A JSR can also be used for this purpose.)

```

.PAGE 0
.
.
LABEL1: . ;PAGE0 SUBROUTINE
.
RET . ;RETURN FROM SUBROUTINE
.
JSRP ADD ;LEGAL CALL TO PAGE 2
.
JSRP SUB ;ILLEGAL CALL TO PAGE 3
.
.PAGE 2 ;START OF PAGE 2 CODE
.
.
ADD: . ;START OF ADD SUBROUTINE
.
RET .
.
JSRP LABEL1 ;ILLEGAL CALL FROM PAGE 2
.
.PAGE 3 ;START OF PAGE 3 CODE
.
.
SUB: . ;SUBTRACT SUBROUTINE
.
RET .
.

```

**7.2.3 Subroutine Pages 2 and 3**

The special characteristics of the JP and JSRP instructions facilitate the use of pages 2 and 3 as subroutine pages. Programmers should consider dedicating these pages to program subroutines for the following reasons:

- A single-byte JSRP can be used to transfer program control to a page 2 subroutine.
- When in page 2 or 3, a single-byte JP can be used to jump to either of these pages.

The following code exemplifies the use of the JP and JSRP instructions to transfer program control to and within pages 2 and 3 as follows. Note that in this example, the ADD subroutine jumps to MEMOVE (Memory Move) routine before returning. Thus, subroutines may share a common "return" subroutine, jumped to and from 2 or 3 with a single-byte JP instruction.

```

.PAGE 0
.
.
JSRP ADD ;CALL ADD SUBROUTINE
.
.
.PAGE 2 ;START OF PAGE 2 CODE
.
.
ADD: . ;ADD SUBROUTINE
.
.
JP MEMOVE ;JUMP TO MEMOVE
. ;"RETURN" ROUTINE (NO
. ;"PUSH" OF STACK)
.PAGE 3 ;START OF PAGE 3 CODE
.
.
MEMOVE: . ;MEMORY MOVE ROUTINE
.
.
RET . ;RETURN TO MAIN PROGRAM
. ;(POP STACK)
.

```

**7.2.4 JID Instruction**

The JID (Jump Indirect) instruction is another page-oriented instruction. JID is an indirect ROM addressing instruction which transfers program control to a new ROM location based upon the contents of a ROM "pointer." The paging features and restrictions associated with the JID instruction are as follows:

1. JID first looks up a ROM pointer based on the contents of A and RAM.
2. JID then transfers program control to the ROM word specified by the contents of the ROM pointer.
3. The ROM pointer and the indirect address jumped to must be within the same 4-page ROM "block" as the JID instruction. Specifically, for purposes of this instruction, the 16 or 32 (for chip 440/444/445) pages of ROM are divided into four or eight blocks as follows:

Block	Pages
1	0-3
2	4-7
3	8-11
4	12-15
5	16-19
6	20-23
7	24-27
8	28-31

For example, if the JID instruction is located in page 5, the ROM pointer and the indirect address to which program control is transferred must be within block 2 (pages 4-7).

### 7.2.5 LQID Instruction

The LQID instruction is an indirect data output instruction. It loads the 8-bit Q register with the 8-bit contents of a particular ROM location pointed to by A and RAM. The paging restrictions associated with this instruction are similar to those associated with the JID instruction, as follows:

1. For purposes of the LQID instruction, as with the JID instruction, ROM is divided into 4-page (or 8-page for chip 440/444/445) ROM "blocks."
2. The ROM location containing the LQID "lookup" data must be within the same ROM block as the LQID instruction.

For example, a LQID instruction located in page 0 must access ROM data located in pages 8 through 11.

### 7.2.6 Restrictions on JP, JSRP, JID, and LQID Instructions

As already mentioned, the ROM address register (P) increments its value when executing an instruction to point to the next memory instruction, automatically "rolling over" to the next page after executing an instruction located in the last word of a page. It is important to realize, however, that P is incremented prior to the execution of the current instruction. This characteristic has important consequences for JP, JSRP, JID and LQID instructions which are located in the last word of a page. Specifically, these instructions operate on the incremented value of P which, because of the increment before execution COP feature, point to the first word of the next page. Consequently, if any of these instructions are placed in the last word of a page, the program treats them as residing on the first word of the following page. Given the paging restrictions associated with these instructions, the following operations and restrictions are associated with the following placements of these instructions:

1. A JP in the last word of a page will go to any location in the following page (except the last word). A JP in the last word of page 1 will be able to go to any location (except the last word) of page 2 or 3 since it is treated as a JP in page 2. Furthermore, a JP in the last word of page 3 will not go to a location within page 2 or 3, but, instead, will go to a location within page 4.
2. A JSRP instruction is not allowed to reside in the last word of page 1, since it will be treated as an illegal use of JSRP in page 2. A JSRP in the last word of page 3, however, is allowed, since it will be treated as a JSRP outside of pages 2 or 3, namely in page 4.
3. An LQID or JID instruction located in the last word of the last page of a particular ROM block (last word of page 3, 7, 11 or 15) will lookup data or transfer program control, respectively, to a location within the next 4-page ROM block.

As is evident from the above, these characteristics are not necessarily restrictions, provided the programmer intentionally uses these instructions to operate in the above manner. For example, a JP on the last word of page 1, unlike other page 1 JP instructions, will be able to transfer program control to the 2-page subroutine pages 2 or 3, provided the operand specifies a location within page 2 or 3. Similarly, an LQID or JID located in the last word of the last page of a ROM block will allow data lookups on or indirect program control transfers to locations within the next ROM block, provided the lookup data or address pointers are placed in the appropriate locations within the next ROM block.

## 7.3 Data Memory Allocation and Manipulation

An important step which should occur prior to writing a COPSTM program is the allocation of program data (registers, flags, counters, etc.) to specific areas of program memory (RAM). This process is referred to as "creating a RAM map" and, although the map will undoubtedly change as programming continues, construction of an initial RAM map will make the ensuing programming process significantly easier.

The COP420 series has four data memory registers, numbered 0 through 3, consisting of 16 4-bit digits. Frequently, accessed data should be stored in locations which are able to be pointed to by loading the B register with a single-byte LBI instruction. These locations consist of digit numbers 0 and 9 through 15 in any data memory register. These areas are indicated by the diagonal-lined areas in Figure 7-1. It requires a double-byte LBI instruction to load the B register to access the other digits in data memory registers, thus requiring an extra program memory word. Single-bit flags and digit counters should be located in these diagonal-lined regions since they tend to be frequently accessed in most programs.

The memory reference instructions LD, X, XDS, and XIS allow the programmer to modify the data memory register address without using an LBI instruction. All of these instructions may modify the upper two bits of

		Digit Address (Bd)															
Register Address	(Br)	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0		///	///	///	///	///	///	///	///						///	///	
		///	///	///	///	///	///	///	///						///	///	
		///	///	///	///	///	///	///	///						///	///	
1		///	///	///	///	///	///	///	///						///	///	
		///	///	///	///	///	///	///	///						///	///	
		///	///	///	///	///	///	///	///						///	///	
2		///	///	///	///	///	///	///	///						///	///	
		///	///	///	///	///	///	///	///						///	///	
		///	///	///	///	///	///	///	///						///	///	
3		///	///	///	///	///	///	///	///						///	///	
		///	///	///	///	///	///	///	///						///	///	
		///	///	///	///	///	///	///	///						///	///	

XIS SKIP XDS SKIP

Figure 7-1. COP420 Data Memory Map

B (Br — RAM register select) by specifying an “r” operand field which is EXCLUSIVE-ORed with the current value of Br. This feature allows the programmer to toggle back and forth between any of the four COP420 data memory registers. For example, data located within the data memory locations marked with shaded boxes in Figure 7-1 can be easily swapped back and forth using the LD and X instructions. They can also be added to or subtracted from each other easily.

The automatic data memory digit address increment and decrement features associated with the XIS and XDS instructions and their skip conditions features facilitate the shifting, adding, and subtracting of the contents of data memory. Data that needs to be shifted should be located in adjacent digit locations (for example, the dotted-box locations in Figure 7-1). Data that needs to be added, subtracted, or shifted should be located in areas adjacent to the XIS or XDS skip boundaries. The dotted locations in Figure 7-1 are against the XIS boundary at digit 15. This allows the programmer to take advantage of the skip feature of the XIS instruction.

The following examples illustrate several of the principles discussed above. The notation M(N1,N2) indicates a particular data memory digit M, where N1 = register number and N2 = digit number.

```

;MOVE M(3,0) TO M(1,0)
LBI    3,0      ;3 TO BR; 0 TO BD (SINGLE
              ;BYTE LBI: D=0)
LD     2        ;M(3,0) TO A; 1 TO BR
X      1        ;A TO M(1,0)

;MOVE MEMORY REGISTER 1 TO MEMORY REGISTER 0
;M(1,15)-M(1,0) TO M(0,15)-M(0,0)
LBI    1,15    ;2 TO BR, 15 TO BD (SINGLE
              ;BYTE LBI)
MV1:   LD     1      ;M(1,15) TO A; 0 TO BR
        XDS   1      ;A TO M(0,15); 1 TO BR; BD-1
              ;TO BD; CONTINUE TO MOVE
              ;NEXT LOWER DIGIT UNTIL
              ;BD GOES PAST 0 AND SKIPS
JP     MV1     ;HERE IF NO SKIP
    
```

```

;LEFT SHIFT DOTTED AREAS OF FIGURE 7-1
;0 TO M(0,12), M(0,12) TO M(0,13) TO M(0,14) TO M(0,15) TO A
CLRA   ;0 TO A
LBI    0,12    ;0 TO BR; 12 TO BD
LSHFT  XIS     ;M(0,12) TO A; 0 TO M(0,12)
        JP     LSHFT ;SHIFT NEXT HIGHER DIGIT
              ;UNTIL "BD" GOES PAST 15
              ;AND SKIPS
    
```

7.4 Subroutine Techniques

Any section of program code used repeatedly within the main program should be coded as a subroutine, preferably on "subroutine pages" 2 or 3 for the reasons discussed above. Subroutines are jumped to or "called" by the JSRP or JSR (double byte) instruction, both of which "push" the stack, saving the next memory location address after the subroutine call in the SA subroutine-save register. The other subroutine-save registers are correspondingly pushed. Subroutine nesting on the COP420 series is permitted to three levels, since this device contains three subroutine-save registers.

Subroutines should terminate with a RET or RETSK instruction, both of which "pop" the subroutine stack, with the program return address in SA being placed in the program counter register. The other subroutine-save registers are also popped. The contents of SC, which is the bottom-most subroutine-save register, are retained in SC in addition to being placed in SB.

It is convenient to think of a subroutine as a program module. The programmer should make its interface to the calling program as clearly defined and as simple as possible. The interface (including data memory registers, entry points, etc., used by the subroutine) should be documented fully by comments to the code.

Subroutine examples presented in this chapter often use the double-byte JSR instruction to call subroutines since no restrictions are associated directly with its use. When writing an actual program, programmers should use the more efficient single-byte JSRP instruction as well as the double-page boundaries of subroutine pages 2 and 3 for placement of subroutine code (as discussed previously) for efficient single-byte jumps while in the pages using the JP instruction.

It is often useful to define multiple-entry points for a single subroutine. The successive-skip feature of the LBI instruction often facilitates this technique.

The RETSK instruction allows the programmer to use an alternate return to the main program (skipping the first program instruction encountered upon return) based upon tests or computations made within the subroutine itself.

Example:

```

.PAGE 0
.
.
JSRP   ADD      ;CALL ADD SUBROUTINE
        .        ;RETURN HERE IF RESULT<9
        .        ;RETURN HERE IF RESULT>9
.
.PAGE 2      ;START PAGE 2 CODE
.
.
ADD:   ADD      ;ADD SUBROUTINE — ADDS
        .        ;TWO BCD DIGITS; RESULT
        .        ;TO A
AISC   7        ;OVERFLOW AND SKIP IF
        .        ;(RESULT>9)
RET    .        ;RETURN WITHOUT SKIP
        .        ;(RESULT<9)
RETSK .        ;RETURN THEN SKIP
        .        ;(RESULT>9)

```

**7.5 Timing Considerations**

Programmers must often synchronize programs with external events ("real-time" programming). Such programs must be balanced with respect to the execution times of the various branches taken by the program. To ensure equal execution times, program timing delays are added. There are numerous ways of introducing timing delays, the simplest but least efficient involving the use of NOPs. Obviously, these are appropriate for only the shortest delays.

A counting loop, such as

```

CLRA
AISC 1
JP . -1      ;ADD 1 TO A UNTIL A
CONTINUE:   ;OVERFLOWS*

```

is more efficient for longer delays, but destroys the previous contents of A. Another method is to use a "scratch-pad" counter in data memory using the XAD instruction. For example, assuming the use of a counter in M(3,15):

```

XAD     3,15    ;COUNTER TO A, A TO M
        .        ;(3,15)
AISC    1       ;ADD 1 TO COUNTER
JP      . -1    ;UNTIL IT OVERFLOWS*
CONTINUE :XAD   ;RESTORE A THEN
        .        ;CONTINUE

```

\* Note: the above timing code example shows the use of a special assembler symbol in the operand of the JP instruction. Namely, the operand of the JP instruction, rather than using a program label, references the assembler location counter (which equals the address of the current program address). The "." signifies the assembler location counter and the value of the operand equals the location counter minus the number of memory bytes to the right of the "." sign. Use of the "." location pointer symbol for transfer-of-control instructions facilitates coding in avoiding the need to create unique program labels to reference memory addresses.

Larger delays may be implemented by using multidigit RAM counters. Another technique is calling unrelated subroutines which change registers or memory locations not currently in use or whose net effect on memory is null. An example of the latter technique is illustrated below.

```

JSR   LR03    ;LEFT ROTATE 3 BITS
JSR   LR01    ;LEFT ROTATE 1 MORE BIT

```

This combination of subroutines only affects A, while maintaining integrity of data in the rotated memory digit.

**7.6 Programming Techniques for the COP421 Series, COP410L and COP411L**

**7.6.1 COP421 Series Programming**

Since the COP421 series differs from the COP420 series only in not having the IN3-IN0 inputs, the foregoing programming considerations and examples for the COP420 series are, for the most part, relevant to COP421 series programming. However, due to its lack of IN inputs, the COP421 series does not include the ININ instruction, and its INIL instruction inputs only CKU into A (when CKU is programmed as a general purpose input). The following are the results of these COP421 differences:

1. MICROBUS™ interface programming is not available since IN3-IN0 cannot be mask-programmed as WR, CS, and RD, respectively. Also, G0 cannot be mask-programmed as a "ready" output to facilitate "handshaking" with a host CPU over the MICROBUS bus. The COP421 may still, however, function as a CPU peripheral component, relying on more general programmed I/O techniques.
2. Due to the lack of IN inputs, other bidirectional I/O pins must be used as general purpose input pins when implementing a programmed input operation.
3. A hardware interrupt using IN1 is not possible. (Setting EN1 has no effect on the operation of any COP421.) Any interrupt servicing must be accomplished using software interrupt techniques.
4. A software interrupt cannot rely on the inputting and testing of the IL3 or ILO latches associated with IN3 and IN0 inputs. Software interrupts require that the interrupt signal be tied to one of the nonlatched input pins. As a result, the input interrupt signal



is a 50% duty cycle, 60Hz square wave, it must be tested at least twice every 1/60 second.

**7.6.2 COP410L/COP411L Programming**

Since the COP410L/COP411L, as with the COP421 series, does not have IN inputs, the above programming considerations relating to the COP421 apply as well to COP410L/COP411L programming. Also, since other hardware logic elements are not included in the architecture of the COP410L, the following additional considerations apply to COP410L programming:

1. The COP410L/COP411L has one-half the ROM and RAM of the COP420 series and COP421 series. ROM consists of 512 x 8-bit words, limiting program code to eight pages (pages 0-7). RAM consists of a 32 x 4-bit RAM, organized as four RAM registers (0-3) consisting of eight 4-bit digits (9-15,0). The LBI register reference instruction should contain a "d" field equal to 9-15 or 0, since all LBIs are single-byte instructions, occupying one word in program memory. A field restriction occurs with respect to the memory reference XAD instruction: only an XAD 3,15 instruction is valid, limiting its use to reference a RAM "scratch-pad" digit contained in M(3,15) only.
2. The COP410L/COP411L has two subroutine save registers, SA and SB. Only two levels of subroutine nesting are allowed. The programmer should also realize that since LQID pushes and pops the stack in performing the operation associated with this instruction, only one level of subroutine nesting should be in effect at the time of the execution of this instruction. (Otherwise the second level of previous subroutine nesting will be disrupted; the previous contents of SB will be lost.)

3. Since the COP410L/COP411L does not have an internal divide-by-1024 time-base counter, the SKT instruction is not available. "Real-time" routines, such as 12-hour timekeeping and the like, must rely on external time-base inputs in order to derive a time-base for such routines (e.g., external 50/60 Hz input for time-of-day routines).
4. Certain deleted or altered instructions have already been mentioned; ININ, INIL, and SKT are not available. LBIs must have a "d" field equal to 9-15 or 0, and XAD's operand must equal 3,15. The following instructions have also been deleted from the COP410L/COP411L instruction set. To the right of each of the following deleted instructions, where appropriate, alternative COP410L/COP411L instructions are shown which, when executed in succession, will perform the same or similar operations as the deleted instruction.

Deleted Instructions	Alternative COP410L/COP411L Instructions
LDD	LBI,LD
CASC	COMP, ASC
ADT	AISC 10, NOP
CQMA	INL
OGI	OMG
XABR	
SKT	
ININ	
INIL	

# Sample Programs

Programmers often build a library of basic routines which are useful in numerous applications. This and the following sections provide examples of several such "utility" routines.

type of routine. Note that the routines may be easily modified to perform moves in the opposite direction (e.g., from register 1 to 0) or to include a move from register 1 to 2.

## 8.1 Register Move Routines

It is often necessary to move data from one memory register to another. The following are examples of this

### 8.1.1 Adjacent Memory Move Routine

```
;ADJACENT MEMORY REGISTER MOVE, MULTIPLE ENTRY POINT SUBROUTINE
;MOV0T1: MOVE MEMORY REGISTER 0 TO REGISTER 1 ENTRY POINT
;MOV2T3: MOVE MEMORY REGISTER 2 TO REGISTER 3 ENTRY POINT
;ROUTINE MOVES DIGITS 15 THROUGH 0
;PREVIOUS CONTENTS OF A AND B ARE LOST
MOV0T1:  LBI      0,15      ;POINT TO M(0,15)
MOV2T3:  LBI      2,15      ;NOTE LBI SUCCESSIVE SKIP FEATURE
MOV:     LD        1        ;TRANSFER M TO A; EXCLUSIVE-OR 1 WITH BR
         XDS      1        ;EXCHANGE A WITH M; EXCLUSIVE-OR 1 WITH BR; DECREMENT BD
         MOV      MOV      ;JUMP TO "MOV" IF MORE DIGITS TO MOVE
         JP       JP       ;RETURN WHEN XDS SKIPS (LAST DIGIT MOVED)
         RET
```

### 8.1.2 Data Memory Shift and Rotate Routines

```
;MULTIPLE ENTRY POINT SUBROUTINE TO RIGHT SHIFT MEMORY REGISTER 0, 1, 2, OR 3 ONE DIGIT
POSITION
;ZEROS ARE SHIFTED INTO DIGIT 15
;PREVIOUS CONTENTS OF A AND B ARE LOST
;RSH0: RIGHT SHIFT REGISTER 0 ENTRY POINT
;RSH1: RIGHT SHIFT REGISTER 1 ENTRY POINT
;RSH2: RIGHT SHIFT REGISTER 2 ENTRY POINT
;RSH3: RIGHT SHIFT REGISTER 3 ENTRY POINT
RSH0:   LBI      0,15      ;POINT TO DIGIT 15 IN APPROPRIATE REGISTER
RSH1:   LBI      1,15      ;NOTE LBI SUCCESSIVE SKIP FEATURE
RSH2:   LBI      2,15
RSH3:   LBI      3,15
        CLRA                      ;ZEROS IN FIRST DIGIT (DIGIT 15)
SHIFT:  XDS                      ;SHIFT RIGHT*
        JP       SHFTR           ;CONTINUE UNTIL ENTIRE REGISTER IS SHIFTED
        RET                      ;RETURN WHEN FINISHED ("XDS" SKIPS)
```

\*Note: The above routine can shift the registers one digit to the left using the "XIS" instruction in place of "XDS" and starting at digit 0.

```
;MULTIPLE ENTRY POINT SUBROUTINE TO LEFT SHIFT THE BITS OF A MEMORY DIGIT
;UPON ENTRY, B MUST POINT TO THE DIGIT TO BE SHIFTED
;ZEROS ARE SHIFTED IN FROM THE RIGHT
;PREVIOUS CONTENTS OF A ARE LOST
;LEF1: SHIFT DIGIT LEFT 1 BIT ENTRY POINT
;LEF2: SHIFT DIGIT LEFT 2 BITS ENTRY POINT
;LEF3: SHIFT DIGIT LEFT 3 BITS ENTRY POINT
LEF3:   LD        X          ;DIGIT TO A
        ADD      X          ;ADD DIGIT TO ITSELF
        X        X          ;SHIFTED DIGIT TO MEMORY
LEF2:   LD        X
        ADD      X
        X        X
LEF1:   LD        X
        ADD      X
        X        X
        RET
```



```

;SUBROUTINE TO DO UNSIGNED BCD INTEGER ADD OF R1 AND R0, RESULT TO R0;
;EACH INTEGER OCCUPIES MEMORY DIGITS 0 (LOW ORDER) THROUGH 12 (HIGH ORDER)
;ON RETURN, C=1 INDICATES OVERFLOW
;PREVIOUS CONTENTS OF A AND B ARE LOST
;ENTRY POINT: BCDADD

```

```

BCDADD:  LBI    1,0      ;POINT TO LOW ORDER DIGIT, REGISTER 1
          RC          ;INITIALIZE C TO "0" (NO CARRY)
ADDL:   LD      1      ;MOVE R1 DIGIT TO A, POINT TO SAME DIGIT IN R0
          AISC    6      ;ADD BCD CORRECTIVE FACTOR OF 6 TO A
          ASC     6      ;RESTORE BCD VALUE IF BCD CORRECTION NOT NECESSARY
          XIS    1      ;MOVE SUM DIGIT TO R0: POINT TO R1, NEXT HIGHER DIGIT
          CBA          ;BD TO A
          AISC    3      ;LAST DIGITS ADDED?
          JP     ADDL   ;NO, ADD NEXT HIGHER DIGITS
          RET          ;YES, RETURN

```

```

;SUBROUTINE TO DO UNSIGNED BCD INTEGER SUBTRACT
;MINUEND IS IN R0, SUBTRAHEND IS IN R1
;DIFFERENCE IS PLACED IN R0
;MINUEND, SUBTRAHEND AND DIFFERENCE DIGITS EACH OCCUPY MEMORY DIGITS 0 (LOW ORDER)
  THROUGH 12 (HIGH ORDER)
;ON RETURN: C = 1 INDICATES NO BORROW, C = 0 INDICATES BORROW
;PREVIOUS CONTENTS OF A AND B ARE LOST
;ENTRY POINT:BCDSUB

```

```

BCDSUB:  LBI    1,0      ;POINT TO LOW ORDER DIGIT IN R1
          SC          ;INITIALIZE C TO "1" (NO BORROW)
SUB:     LD      1      ;LOAD R1 DIGIT TO A, POINT TO SAME DIGIT IN R0
          CASC       ;SUBTRACT R1 DIGIT FROM R0 DIGIT
          ADT        ;BCD ADJUST IF BORROW (C=0)
          XIS    1      ;PLACE DIFFERENCE DIGIT IN R0, POINT TO NEXT HIGHER
                        DIGIT IN R1
          CBA          ;BD TO A
          AISC    3      ;HIGH ORDER DIGITS (12) SUBTRACTED?
          JP     SUB   ;NO, SUBTRACT NEXT HIGHER DIGITS
          RET          ;YES, RETURN

```

### 8.2.2 BCD Integer Multiply Routine

This routine multiplies the contents of data memory register 2 with register 1, placing the result in register 2 (digits 0-12). It also calls the BCD add routine ("BCDADD") given above. Note that a loop-counter is

contained in M(0,13) which causes the program to return after all twelve digits have been multiplied. Note the alternate-return feature of page 3 subroutine TMZERO (Test Memory Digit = 0).

```

;TWO-LEVEL BCD INTEGER MULTIPLY SUBROUTINE
;12 DIGIT BCD INTEGER CONTAINED IN REGISTER 1, DIGITS 0-12 (LOW ORDER TO HIGH ORDER)
  MULTIPLIED BY 12 DIGIT BCD
;INTEGER CONTAINED IN REGISTER 2, DIGITS 0-12 (LOW ORDER TO HIGH ORDER), RESULT TO REGISTER 2
;MULTIPLICATION OF DIGITS PERFORMED BY MULTIPLE ADDITIONS OF REGISTER 1 ACCORDING TO
  VALUE OF REGISTER 2
;DIGITS
;DIGIT ADDITION RESULTS TEMPORARILY STORED IN R0 AND CONSECUTIVELY RIGHT SHIFTED INTO
  RESULT REGISTER 2, HIGH ORDER DIGIT
;ENTRY POINT: MULT
;SUBROUTINES CALLED: RSHR0, RSHR2, CLR, DEC1, INC1, TMZERO, BCDADD

```

```

MULT:    LBI    0,13    ;POINT TO M(0,13)
          JSR    CLR    ;CLEAR REGISTER 0 DIGITS 13-0
MULT1:   LBI    2,0     ;POINT TO M(2,0)
          JSR    TMZERO ;IS M(2,0) = 0?
          JP     NOTZ   ;NO, JUMP TO NOTZ
          JSR    RSHR0  ;YES, RIGHT SHIFT REGISTER 0, DIGITS 12-0
          JSR    RSHR2  ;RIGHT SHIFT REGISTER 2, DIGITS 12-0
          LBI    0,13    ;POINT TO LOOP COUNTER
          LD     A      ;LOOP COUNTER TO A
          AISC    3      ;IS COUNTER > 12?
          JP     .+2    ;NO, CONTINUE
          RET          ;YES, ALL DIGITS MULTIPLIED, RETURN

```



```

                JSR      INC1      ;CONTINUE, INCREMENT LOOP COUNTER DIGIT
                JP       MULT1     ;MULTIPLY NEXT HIGHER ORDER DIGITS
NOTZ:          JSR      DEC1      ;DECREMENT M(2,0)
                JSR      BCDADD    ;ADD R0, DIGITS 0-12, TO R1, DIGITS 0-12, RESULT TO R0
                JP       MULT1     ;JUMP BACK TO MULT1

```

```

;MULTIPLE ENTRY POINT SUBROUTINE TO RIGHT SHIFT DIGITS 12-0 OF REGISTER 2
;ON RETURN A CONTAINS LOW ORDER REGISTER DIGIT
;RSHR0: RIGHT SHIFT DIGITS OF REGISTER 0 ENTRY POINT
;RSHR2: RIGHT SHIFT DIGITS OF REGISTER 2 ENTRY POINT

```

```

RSHR0:  LBI      0,12      ;POINT TO HIGH ORDER DIGIT, REGISTER 0
RSHR2:  LBI      2,12      ;POINT TO HIGH ORDER DIGIT, REGISTER 2
RSH:    XDS
                JP       RSH
                RET

```

```

;SUBROUTINE TO CLEAR ALL DIGITS TO THE RIGHT AND INCLUSIVE OF A HIGH ORDER DIGIT OF A REGISTER
;ON ENTRY, B MUST POINT TO THE REGISTER AND HIGH ORDER DIGIT NUMBER

```

```

CLR:    CLRA
                XDS
                JP       CLR      ;CLEAR REGISTER, STARTING WITH HIGH ORDER DIGIT
                RET              ;RETURN WHEN DIGIT 0 CLEARED

```

```

;MULTIPLE ENTRY SUBROUTINE TO EITHER DECREMENT OR INCREMENT BY 1 THE VALUE OF A
MEMORY DIGIT

```

```

;ON ENTRY, B MUST POINT TO THE DIGIT TO BE OPERATED UPON
;DEC1: ENTRY POINT TO DECREMENT A DIGIT
;INC1: ENTRY POINT TO INCREMENT A DIGIT

```

```

DEC1:   CLRA          ;15 TO A
                COMP      ;ADD MEMORY DIGIT TO A
ADEX:   ADD           ;EXCHANGE BACK TO MEMORY
                X
                RET
INC1:   CLRA
                AISC      1      ;1 TO A
                JP       ADEX    ;ADD AND EXCHANGE WITH MEMORY DIGIT

```

```

;SUBROUTINE TO TEST MEMORY DIGIT EQUAL TO ZERO
;ON ENTRY, B MUST POINT TO MEMORY DIGIT TO BE TESTED
;ON RETURN, SKIP FIRST INSTRUCTION IF MEMORY DIGIT EQUAL TO ZERO
;NORMAL RETURN IF MEMORY DIGIT NOT EQUAL TO ZERO

```

```

TMZERO: CLRA          ;0 TO A
                SKE       ;DIGIT = ZERO?
                RET       ;NO, NORMAL RETURN
                RETSK     ;YES, RETURN THEN SKIP

```

### 8.3 Simple Display Loop Routine

The following routine is a simple LED display loop routine. It illustrates the use of LEI and LQID instructions, both designed to facilitate the outputting of segment data to a multiplexed display. Setting bit 2 of the EN register enables Q latch (segment) data to the L I/O ports; resetting EN2 disables the L I/O ports, providing segment blanking for the LED display. EN2 is set and reset by the LEI4 and LEI0 instructions, respectively.

LQID loads the 8-bit Q register with the contents of a ROM location pointed to by A and M (ROM "lookup" data must be within the same 4-page ROM block as the LQID instruction). In this example, since A is always equal to zero at the time of the LQID instruction, the ROM data accessed by this instruction must be within the first 16 words of the first page of the ROM block in which the LQID instruction is located as pointed to by

the 4-bit contents of M (P9 and P8 remain the same, P7-P4 equal zero). For example, if, as is the case for the following routine, LQID is in page 5, it will lookup data within one of the first 16 locations of page 4. The value of the contents of the memory digit pointed to by the B register at the time of the LQID instruction determines which one of the 16 words is accessed (e.g., if M = 2, word 2 is loaded into Q).

Due to these considerations, page 4, words 0-9 should equal the 8-bit, 7-segment decode lookup data for the BD digits 0-9 respectively. In this example the low order bit (decimal point) of each lookup data word is reset, signifying that the decimal point is off.) ROM 7-segment decode lookup data is placed in ROM memory locations by the assembler .WORD directive.

Another feature of this routine is the dual function of Bd. Its value may be output directly to the D outputs to select one of 16 digits of the multiplexed display (assuming the D outputs are connected to a 1-of-16 decoder/driver device). Also, its value is used to select one of 16 RAM digits whose contents are used by the LQID instruction to access the segment data to be output to the selected digit. To facilitate coding (by avoiding the need to change the value of Bd after its

contents are output to D to select or display digit), RAM digit locations should correspond to the digit of the display. In other words, RAM digits 0-15 should contain, respectively, the LQID pointers to segment data for display digits 0-15. This technique, used below, allows Bd to first enable the appropriate display digit and then, without its value being changed, to point to the RAM digit used to access the segment data for the same display digit.

```

;SEVEN SEGMENT DECODE DATA TABLE:
;ROM BITS 17-10 = SA-SG,D.P.(DECIMAL POINT) BITS, RESPECTIVELY
LOOKUP:  .PAGE      4      ;PLACE LOOKUP DATA IN WORDS 0-9, PAGE 4
          .WORD     X'FC    ;= 0 (SEVEN SEGMENT DECODE HEX VALUES)
          .WORD     X'60    ;= 1
          .WORD     X'DA    ;= 2
          .WORD     X'F2    ;= 3
          .WORD     X'66    ;= 4
          .WORD     X'B6    ;= 5
          .WORD     X'BE    ;= 6
          .WORD     X'E0    ;= 7
          .WORD     X'F4    ;= 8
          .WORD     X'F6    ;= 9
          ;NEXT FIVE LOCATIONS CAN BE USED FOR SPECIAL
          ;ALPHABETICAL DISPLAY
          ;CHARACTER DATA

;BEGIN CODE FOR DISPLAY LOOP
DSPLY:   .PAGE      5      ;PLACE FOLLOWING CODE ON PAGE 5
LOOP:    LBI        0,15   ;POINT TO HIGH ORDER RAM DIGIT, BD = 15
          CLRA      ;A = 0 FOR LOOKUP
          LEI        0      ;BLANK SEGMENTS (EN2 = 0)
          OBD       ;OUTPUT DIGIT VALUE
          LQID      ;LOOKUP DATA TO Q
          LEI        4      ;OUTPUT SEGMENT DATA (EN2 = 1)
          CBA       ;BD TO A
          AISC      15     ;DECREMENT A
          JP        .+3    ;JUMP 3 WORDS WHEN FINISHED
          CAB       ;A(BD-1) TO BD
          JP        LOOP   ;DISPLAY NEXT LOWER DIGIT
          .          ;CONTINUE WHEN FINISHED

```

#### 8.4 Interrupt Service Routine

Setting bit 1 of the EN register enables the COP420 series and COP444L IN1 input as an interrupt input, responding to low-going pulses. Upon the occurrence of an interrupt signal, the subroutine stack is pushed and program control is transferred to the last word of page 3 (address OFF<sub>16</sub>). The following routine contains code which may be placed at the beginning and end of the interrupt service routine to save the contents of A, C

and B, freeing them for use by the interrupt routine. At the end of the routine, the previous contents of A, C and B are restored for use by the main program. It should be noted that the main program need only enable IN1 as an interrupt input once; thereafter, the interrupt service routine, itself, re-enables interrupt servicing (LEI1 instruction before return).

```

;INTERRUPT SERVICE ROUTINE TO SAVE AND RESTORE THE CONTENTS OF A, C, AND B (BR AND BD)
;IN MEMORY REGISTER 0, DIGITS 0-2.
;AUTOMATIC ENTRY TO LAST WORD OF PAGE 3
;ON RETURN, IN1 INPUT RE-ENABLED AS INTERRUPT INPUT
INTSER:  NOP          ;FIRST INTERRUPT ROUTINE INSTRUCTION MUST BE A NOP
          ;(LOCATION X'FF)
          XAD        0,0  ;SAVE IN M(0,0)
          CBA        ;BD TO A
          XAD        0,1  ;SAVE BD IN M(0,1)
          XABR       ;BR TO A
          SKC        ;CARRY = 1?
          AISC       8    ;NO, SET A3

```

```

XAD      0,2      ;SAVE C AND BR IN M(0,2)
           .
           .
           .
LDD      0,2      ;M(0,2) (C AND BR) TO A
RC       .
           ;RESET CARRY
AISC     8        ;A3 SET (SAVED CARRY = 0?)
SC       .
           ;NO RESTORE CARRY = 1
XABR    .
LDD      0,1      ;M(0,1) (BD) TO A
CAB      .
           ;RESTORE BD
LDD      0,0      ;M(0,0) TO A, RESTORE A
LEI      1        ;ENABLE INTERRUPT (SET IN1)
RET      .
           ;RETURN FROM INTERRUPT SERVICE ROUTINE

```

## 8.5 Timekeeping Routine

The following multilevel subroutine counts time in a 12-hour format. It relies on the COP420 system oscillator itself (controlled by an inexpensive 3.58 MHz color TV crystal), and the COP420 internal time-base counter for a real-time base, rather than on a 60 Hz external input. The subroutine is entered each time the SKT instruction skips, indicating time-base counter overflow. Overflow frequency is dependent upon the frequency of the COPS™ system oscillator. This frequency equals the oscillator frequency, first divided by 16 by the instruction cycle divider, then by 1024 by the internal 10-bit time-base counter. In this case, the SKT overflow frequency will equal a fractional number: 218.478 Hz (3.58 MHz divided by 16, divided by 1024). Consequently, the timekeeping calling routine must execute an SKT instruction at least once approximately each 218 Hz to ensure that each SKT overflow is detected.

As indicated above, using an inexpensive TV crystal results in a fractional SKT frequency. Program compensation techniques, therefore, must be employed to derive an integer which may be used by the program in counting seconds, the basic timekeeping units. This routine derives this integer and uses it to keep accurate time in the following manner:

1. A 2-digit binary "SKT" counter in RAM is initialized to different values at different times during the course of an hour so that the total counts for the hour equal an integer which corresponds to the 218.478 Hz SKT frequency.
2. Every odd second in the range of 0-59 seconds, the SKT counter is set to 218, decremented by one each time the SKT instruction skips. When decremented to 0, a 2-digit BCD "seconds" counter in RAM is incremented by 1. (The seconds counter overflows every 60 counts to a 2-digit BCD "minute" counter. The minutes counter overflows every 60 counts to a 1-digit "hours" counter.)
3. Every even second in the range of 0-59 minutes, the SKT counter is set to 218 and decremented as above.
4. Every hour, the SKT counter is set to 199 and decremented as above.

These compensation techniques result in a timekeeping routine which is accurate at the end of each hour. (During the hour, inaccuracy is extremely small.) The basis for the preceding compensation scheme is as follows:

1. Using a 3.58 MHz crystal resulting in a 218.478 Hz SKT frequency, an SKT integer count of 786,521 is obtained each hour (218.478 × 3600 seconds/hours).
2. Using the this compensation scheme, the same number of "SKT" counts (786,521) is required to increment the time by one hour. This follows since 392,400 counts are required by the "odd" seconds compensation (30 × 60 × 218 counts); 381,060 by the "even" seconds compensation (29 × 60 × 219 counts); 12,862 by the "minutes" compensation (59 × 218 counts) and 199 by the "hours" compensation — resulting in a total hours count of 786,521.

This subroutine is coded to reside on subroutine page 2. The source code provided on the following page also illustrates the use of the STARPLEX™ assembler .LOCAL directive and local symbol labels. Specifically, the program begins and ends with a .LOCAL directive, making the memory addresses between them a local region. Within this local region, local symbols (labels whose first character is a "\$") will be defined only within the local region; they will not conflict with labels appearing in other portions of program source code. This relieves the programmer from worry about duplicate label definitions, allowing the subroutine or other utility program to be included or added to different programs, regardless of the labels used by these other programs. In effect, therefore, utility programs or commonly used subroutines may be coded in this manner and placed in separate "utility" files on a disk. They can then be added or included, when needed, to main programs at a later date. For an example of a program which includes this "TIMEKP" subroutine (using the assembler .INCLD directive), see the following program.

Local symbols must begin with a "\$" and be unique within the particular local region in the first four characters following the "\$." The programmer may, as is done in this example, use local labels with more than four characters for convenience and, although not "recognized" by the assembler, these extra characters

will be printed out on the assembler output listing. Note: The label of the starting address of a local utility routine must be a long (regular) label since it will be referenced by a portion of the program outside of the local region (e.g., "TIMEKP" is not a local label).

```
;PAGE 2 SUBROUTINE TO KEEP TIME IN A 12-HOUR FORMAT USING A 3.58 MHz TV CRYSTAL
;2-DIGIT "SKT" COUNTER CONTAINED IN M(2,15)-M(2,14): HIGH TO LOW ORDER
;1-DIGIT BINARY HOURS COUNTER IN M(2,13)
;2-DIGIT BCD MINUTES COUNTER IN M(2,12)-M(2,11): HIGH TO LOW ORDER
;2-DIGIT BCD SECONDS COUNTER IN M(2,10)-M(2,9): HIGH TO LOW ORDER
;ENTRY POINT: TIMEKP; ENTRY UPON SKT INSTRUCTION OVERFLOW
;SUBROUTINES CALLED: INC2
```

```

.PAGE      2          ;PAGE 2 SUBROUTINE
.LOCAL     ;CREATE LOCAL REGION FOR LOCAL SYMBOLS
$COUNT   =2,14     ;ASSIGN "COUNT" = ADDRESS OF LOW ORDER SKT
                    ;COUNTER DIGIT
TIMEKP:    LBI       $COUNT ;POINT TO LOW ORDER DIGIT OF SKT COUNTER
           LD        ;LOAD DIGIT TO A
           AISC     15      ;DIGIT = 0? (A = DIGIT - 1)
           JP       $HIGHST ;YES, TEST HIGH ORDER DIGIT
           X        ;NO, EXCHANGE DIGIT -1 INTO M
           RET      ;RETURN UNTIL NEXT SKT OVERFLOW
$HIGHTST:  XIS      ;REPLACE DIGIT IN COUNTER, INCREMENT BD
           JP       TIMEKP+1 ;JUMP BACK AND TEST HIGH ORDER DIGIT — IF ALREADY
                    ;TESTED AND = 0. SKIP AND CONTINUE
           LBI     $SECS   ;POINT TO LOW ORDER SECS DIGIT
           JSR     $INC2   ;INCREMENT SECS COUNTER
           JP     $TSEC    ;SECS<60, TEST SECS FOR ODD OR EVEN
           STII    0       ;SECS = 60, 0 TO HIGH ORDER DIGIT, POINT TO LOW-ORDER
                    ;MINS DIGIT
           JSR     $INC2   ;INCREMENT MINS COUNTER
           JP     $C218    ;MINS<60, SET COUNTER = 218
           STII    0       ;MINS = 60, 0 TO HIGH ORDER DIGIT, POINT TO HOURS DIGIT
           LD      ;LOAD HOURS DIGIT TO A
           AISC     1       ;INCREMENT HOURS
           X        ;PLACE IN M. PREVIOUS HRS TO A
           AISC     4       ;HOURS>12?
           JP     $C199    ;NO, SET COUNTER = 199
           STIII   1       ;YES, SET HOURS = 1
SC199:     LBI     $COUNT ;POINT TO LOW ORDER COUNTER DIGIT
           STII    7       ;SET COUNTER = 199 (BINARY 12,7)
           STII    12      ;
           RET      ;RETURN UNTIL NEXT SKT OVERFLOW
STSEC:     LBI     $SECS   ;POINT TO LOW ORDER SECS DIGIT
           SKMBZ   1       ;SECS ODD?
           JP     $C218    ;YES, SET COUNTER = 218 (BINARY 13,10)
$C219:     LBI     $COUNT ;NO, POINT TO LOW ORDER COUNTER DIGIT
           STII    11      ;SET COUNTER = 219 (BINARY 13,11)
$C218:     LBI     COUNT  ;POINT TO LOW ORDER COUNTER DIGIT
           STII    10      ;SET COUNTER = 218
           JP     $C21X    ;JUMP TO "C21X" THEN RETURN
```

```
;SUBROUTINE TO INCREMENT A 2-DIGIT BCD RAM COUNTER
;ON ENTRY, B MUST POINT TO LOW ORDER DIGIT OR COUNTER
;ENTRY POINT: INC2
;NORMAL RETURN IF 2 DIGIT VALUE LESS THAN 60
;RETURN THEN SKIP IF 2 DIGIT VALUE EQUAL TO 60
;BOTH RETURNS EXIT WITH B POINTING TO HIGH ORDER DIGIT
```

```

INC2:   SC           ;INITIALIZE C TO 1 TO ADD TO LOW ORDER DIGIT
        CLRA        6 ;ZERO TO A
        AISC        ;BCD ADJUST RESULT IF NECESSARY
        ASC         ;IF RESULT>9, LOW ORDER DIGIT = 0
        ADT
        XIS
        CLRA
        AISC        6 ;PLACE INCREMENTED DIGIT IN M, POINT TO HIGH ORDER DIGIT
        ;ZERO TO A
        ;ADD CARRY, IF PROPAGATED FROM LOW ORDER DIGIT TO
        ;HIGH ORDER DIGIT
        AISC
        ADT         ;BCD RESULT IF NECESSARY
        X           ;REPLACE DIGIT IN M
        LD          ;LOAD HIGH ORDER DIGIT INTO A
        AISC        10 ;HIGH ORDER DIGIT — 6 (COUNT = 60)?
        RET         ;NO, NORMAL RETURN
        REGSK       ;YES, RETURN THEN SKIP
        .LOCAL      ;END LOCAL REGION
    
```

### 8.6 String Search Routine

It is often necessary to search data memory for a string of characters. The following routine searches register 0 for a match with three contiguous 4-bit characters, "X," "Y," and "Z." Note that a match with more than three characters is easily accommodated by providing for additional character tests, using the simple

character test instruction provided below containing modified LDD instructions whose operands specify the additional characters to be matched. Also, the code may be easily modified to search through more than one RAM register for a match.

```

;SUBROUTINE TO SEARCH STRING OF DATA MEMORY CHARACTERS FOR A MATCH WITH
;"X," "Y," AND "Z" CONTIGUOUS CHARACTERS
;16 4-BIT CHARACTERS ASSUMED STORED IN M(0,15) THROUGH M(0,0)
;"X," "Y," AND "Z" CHARACTERS ASSUMED STORED IN AND ASSIGNED VALUES OF M(1,15)
;THROUGH M(1,13), RESPECTIVELY
;NORMAL RETURN IF NO MATCH
;RETURN THEN SKIP IF MATCH OCCURS WITH THE ACCUMULATOR CONTAINING THE DIGIT
;NUMBER OF "X"
        X = 1,15
        Y = 1,14
        Z = 1,13
SEARCH: LBI         0,15 ;POINT TO M(0,15)
LOOKX:  LDD         X    ;X TO A
        SKE        ;X FOUND?
        JP         NOX  ;NO, JUMP TO X
        XDS        ;YES, POINT TO NEXT LOWER DIGIT
        JP         LOOKY ;LOOK FOR Y MATCH, IF AT M(0,0) SKIP AND
        ;NORMAL RETURN — NO MATCH
NOX:    LD          ;DECREMENT DIGIT POINTER
        XDS
        JP         ;LOOP AGAIN FOR X MATCH, IF AT M(0,0), SKIP AND
        ;NORMAL RETURN — NO MATCH
LOOKY:  RET
        LDD         Y    ;Y TO A
        SKE        ;Y FOUND?
        JP         LOOKX ;NO, TRY AGAIN
        XDS        ;YES, POINT TO NEXT LOWER DIGIT
        P          LOOKX ;LOOK FOR Z MATCH, IF AT M(0,0), SKIP AND
        ;NORMAL RETURN — NO MATCH
LOOKZ:  LDD         Z    ;Z TO A
        SKE        ;Z FOUND?
        JP         LOOKX ;NO, TRY AGAIN
        OBA        ;YES, MATCH COMPLETE, COPY Z DIGIT ADDRESS TO A
        AISC        2   ;ADD 2 TO A TO EQUAL X DIGIT ADDRESS
        RETSK      ;RETURN THEN SKIP — MATCH FOUND
    
```

# Appendix A

## ASCII Character Set

Hex.	Char.	Hex.	Char.	Hex.	Char.	Hex.	Char.	Hex.	Char.
00	NUL	1A	SUB	34	4	4E	N	68	h
01	SOH	1B	ESC	35	5	4F	O	69	i
02	STX	1C	FS	36	6	50	P	6A	j
03	ETX	1D	GS	37	7	51	Q	6B	k
04	EOT	1E	RS	38	8	52	R	6C	l
05	ENO	1F	US	39	9	53	S	6D	m
06	ACK	20	SP	3A	:	54	T	6E	n
07	BEL	21	!	3B	;	55	U	6F	o
08	BS	22	"	3C	<	56	V	70	p
09	HT	23	#	3D	=	57	W	71	q
0A	LF	24	\$	3E	>	58	X	72	r
0B	VT	25	%	3F	?	59	Y	73	s
0C	FF	26	&	40	@	5A	Z	74	t
0D	CR	27	'	41	A	5B	[	75	u
0E	SO	28	(	42	B	5C	\	76	v
0F	SI	29	)	43	C	5D	]	77	w
10	DLE	2A	*	44	D	5E	^	78	x
11	DC1	2B	+	45	E	5F	-	79	y
12	DC2	2C	,	46	F	60		7A	z
13	DC3	2D	-	47	G	61	a	7B	{
14	DC4	2E	.	48	H	62	b	7C	}
15	NAK	2F	/	49	I	63	c	7D	~
16	SYN	30	0	4A	J	64	d	7E	
17	ETB	31	1	4B	K	65	e	7F	DEL
18	CAN	32	2	4C	L	66	f		
19	EM	33	3	4D	M	67	g		

### Definitions for Non-Printing Characters

Character	Definition	Character	Definition
NUL	NULL	SO	SHIFT OUT
SOH	START OF READING; ALSO START OF MESSAGE	SI	SHIFT IN
STX	START OF TEXT; ALSO EOA, END OF ADDRESS	DLE	DATA LINK ESCAPE
ETX	END OF TEXT; ALSO EOM, END OF MESSAGE	DC1	DEVICE CONTROL 1
EOT	END OF TRANSMISSION (END)	DC2	DEVICE CONTROL 2
ENQ	ENQUIRY (ENQRY); ALSO WRU	DC3	DEVICE CONTROL 3
ACK	ACKNOWLEDGE. ALSO RU	DC4	DEVICE CONTROL 4
BEL	RINGS THE BELL	NAK	NEGATIVE ACKNOWLEDGE
BS	BACKSPACE	SYN	SYNCHRONOUS IDLE (SYNC)
HT	HORIZONTAL TAB	ETB	END OF TRANSMISSION BLOCK
LF	LINE FEED OR LINE SPACE (NEW LINE); ADVANCES PAPER TO NEXT LINE BEGINNING OF LINE	CAN	CANCEL (CANCL)
VT	VERTICAL TAB (VTAB)	EM	END OF MEDIUM
FF	FORM FEED TO TOP OF NEXT PAGE (PAGE)	SUB	SUBSTITUTE
CR	CARRIAGE RETURN	ESC	ESCAPE. PREFIX
		FS	FILE SEPARATOR
		GS	GROUP SEPARATOR
		RS	RECORD SEPARATOR
		US	UNIT SEPARATOR
		SP	SPACE

## Appendix B

## Alphabetical Mnemonic Index of ASMCOP Instructions

Instruction	Hexadecimal Op Code	Description
ADD	31	Add A to RAM
ADT	4A	Add Ten to A
AISC 1-15	51-5F	Add Immediate, Skip on Carry
ASC	30	Add with Carry, Skip on Carry
CAB	50	Copy A to Bd
CAMQ*	33/3C	Copy A, RAM to Q
CASC	10	Complement and Add with Carry, Skip on Carry
CBA	4E	Copy Bd to A
CLRA	0	Clear A
COMP	40	Ones complement of A to A
CQMA*	33/2C	Copy A to RAM, A
ING*	33/2A	Input G Ports to A
INIL*	33/0	Input IL Latches to A
ININ	33/28	Input IN Inputs to A
INL*	33/2E	Input L Ports to M, A
IT*	33/39	Idle Till Time Overflow
JID	FF	Jump Indirect
JMP*	60-67/0-FF	Jump
JP	80-BE,CO-CD	Jump within Page
JSR*	68-6F/0-FF	Jump to Subroutine
JSRP	80-BE	Jump to Subroutine Page
LBI 0,9-15,0	8-F	Load B Immediate (Single-byte)
LBI 1,9-15,0	18-1F	
LBI 2,9-15,0	28-2F	
LBI 3,9-15,0	38-3F	
LBI* 0,1-8	33/81-88	Load B Immediate (Double-byte)
LBI* 1,1-8	33/91-98	
LBI* 2,1-8	33/A1-AB	
LBI* 3,1-8	33/B1-B8	
LD 0,1,2,3	5,15,25,35	Load RAM into A
LDD* 0-7,0-15	23/0-7F	Load A with RAM
LEI* 0-15	33/60-6F	Load EN Immediate
LQID	BF	Load Q Indirect
NOP	44	No Operation
OBD*	33/3E	Output Bd to D Outputs
OGI*	33/50-5F	Output to G Ports Immediate
OMG*	33/3A	Output RAM to G Ports
RC	32	Reset C
RET	48	Return
RETSK	49	Return then Skip
RMB 0,1,2,3	4C,45,42,43	Reset RAM Bit
SC	22	Set C
SMB 0,1,2,3	4D,47,46,48	Set RAM Bit
SKC	20	Skip if C is True
SKE	21	Skip if A Equals RAM Digit
SKGBZ* 0,1,2,3	33/1,11,3,13	Skip if G Bit is Zero
SKGZ*	33/21	Skip if G Equals Zero (All 4-Bits)
SKMBZ 0,1,2,3	1,11,3,13	Skip if RAM Bit is Zero
SKT	41	Skip on Timer
STII	70-7F	Store Memory Immediate and Increment Bd
X 0,1,2,3	6,16,26,36	Exchange RAM with A
XABR	12	Exchange A with Br
XAD* 0-7,0-15	23/80-FF	Exchange RAM with A and Decrement Bd
XIS 0,1,2,3	4,14,24,34	Exchange RAM with A and Increment Bd
XAS	4F	Exchange A with SIO
XOR	2	Exclusive-OR A with RAM

\* Double-byte Instruction

# Appendix C

## Numeric Index of ASMCOP Instructions

Table C-1. COP 420/421 Instructions

00	CLRA	25	LD 2	49	RETSK	68	JSR*** to Page 0, 1, 2, or 3
02	SKMBZ 0	26	X 2	4A	ADT	69	JSR*** to Page 4, 5, 6, or 7
03	SKMBZ 2	28	LBI 2,9	4B	SMB 3	6A	JSR*** to Page 8, 9, 10, or 11
04	XIS 0	29	LBI 2,10	4C	RMB 0	6B	JSR*** to Page 12, 13, 14, or 15
05	LD 0	2A	LBI 2,11	4D	SMB 0	6C	Invalid
06	X 0	2B	LBI 2,12	4E	CBA	6D	Invalid
07	XDS 0	2C	LBI 2,13	4F	XAS	6E	Invalid
08	LBI 0,9	2D	LBI 2,14	50	CAB	6F	Invalid
09	LBI 0,10	2E	LBI 2,15	51	AISC 1	70	STII 0
0A	LBI 0,11	2F	LBI 2,0	52	AISC 2	71	STII 1
0B	LBI 0,12	30	ASC	53	AISC 3	72	STII 2
0C	LBI 0,13	31	ADD	54	AISC 4	73	STII 3
0D	LBI 0,14	32	RC	55	AISC 5	74	STII 4
0E	LBI 0,15	33	Two Word* (except LDD, XAD, JMP, and JSR)	56	AISC 6	75	STII 5
0F	LBI 0,0	34	XIS 3	57	AISC 7	76	STII 6
10	CASC	35	LD 3	58	AISC 8	77	STII 7
11	SKMBZ 1	36	X 3	59	AISC 9	78	STII 8
12	XABR	37	XDS 3	5A	AISC 10	79	STII 9
13	SKMBZ 3	38	LBI 3,9	5B	AISC 11	7A	STII 10
14	XIS 0	39	LBI 3,10	5C	AISC 12	7B	STII 11
15	LD 1	3A	LBI 3,11	5D	AISC 13	7C	STII 12
16	X 1	3B	LBI 3,12	5E	AISC 14	7D	STII 13
18	LBI 0,9	3X	LBI 3,13	5F	AISC 15	7E	STII 14
19	LBI 0,10	3D	LBI 3,14	60	JMP*** to Page 0, 1, 2, or 3	7F	STII 15
1A	LBI 0,11	3E	LBI 3,15	61	JMP*** to Page 4, 5, 6, or 7	80	JSRP to Word xx (0-3F) or JP to Page 2, Word xx (0-3F)
1B	LBI 0,12	3F	LBI 3,0	62	JMP*** to Page 8, 9, 10, or 11	8E	opcode 80 + xx
1C	LBI 0,13	40	COMP	63	JMP*** to Page 12, 13, 14, or 15	8F	LDI
1D	LBI 0,14	41	SKT	64	Invalid	C0	JP to word xx (0-3F)
1E	LBI 0,15	42	RMB 2	65	Invalid	CE	opcode = C0 + xx
1F	LBI 0,0	43	RMB 3	66	Invalid	FF	JID
20	SKC	44	NOP	67	Invalid		
21	CKE	45	RMB 1				
22	SC	46	SMB 2				
23	LDD/XAD**	47	SMB 1				
24	XIS 2	48	RET				

\*\*\* 00 + xx JSR or JMP to page 0, 1, 10, or 14, word xx (0-3F) 00-3F  
 40 + xx JSR or JMP to page 1, 5, 11, or 15, word xx (0-3F) 40-7F  
 80 + xx JSR or JMP to page 2, 6, 12, or 16, word xx (0-3F) 80-BF  
 C0 + xx JSR or JMP to page 3, 7, 13, or 17, word xx (0-3F) C0-FF



# Appendix C

## Numeric Index of ASMCOP Instruction (Continued)

Table C-1. COP 420/421 Instructions — Second Word of Two Word Instructions

00	INIL*	6C	LEI 12	08	LDD 0,8	31	LDD 3,1	9A	XAD 1,10
01	SKGBZ 0	6D	LEI 13	09	LDD 0,9	32	LDD 3,2	9B	XAD 1,11
11	SKGBZ 1	6E	LEI 14	0A	LDD 0,10	33	LDD 3,3	9C	XAD 1,12
03	SKGBZ 2	6F	LEI 15	0B	LDD 0,11	34	LDD 3,4	9D	XAD 1,13
13	SKGBZ 3	81	LBI 0,1	0C	LDD 0,12	35	LDD 3,5	9E	XAD 1,14
21	SKGZ	82	LBI 0,2	0D	LDD 0,13	36	LDD 3,6	9F	XAD 1,15
28	ININ	83	LBI 0,3	0E	LDD 0,14	37	LDD 3,7	A0	XAD 2,0
2A	ING	84	LBI 0,4	0F	LDD 0,15	38	LDD 3,8	A1	XAD 2,1
2C	CQMA	85	LBI 0,5	10	LDD 1,0	39	LDD 3,9	A2	XAD 2,2
2E	INL	86	LBI 0,6	11	LDD 1,1	3A	LDD 3,10	A3	XAD 2,3
3A	OMG	87	LBI 0,7	12	LDD 1,2	3B	LDD 3,11	A4	XAD 2,4
3C	CAMQ	88	LBI 0,8	13	LDD 1,3	3C	LDD 3,12	A5	XAD 2,5
3E	OBD	91	LBI 1,1	14	LDD 1,4	3D	LDD 3,13	A6	XAD 2,6
50	OGI 0	92	LBI 1,2	15	LDD 1,5	3E	LDD 3,14	A7	XAD 2,7
51	OGI 1	93	LBI 1,3	16	LDD 1,6	3F	LDD 2,15	A8	XAD 2,8
52	OGI 2	94	LBI 1,4	17	LDD 1,7	80	XAD 0,0	A9	XAD 2,9
53	OGI 3	95	LBI 1,5	18	LDD 1,8	81	XAD 0,1	AA	XAD 2,10
54	OGI 4	96	LBI 1,6	19	LDD 1,9	82	XAD 0,2	AB	XAD 2,11
55	OGI 5	97	LBI 1,7	1A	LDD 1,10	83	XAD 0,3	AC	XAD 2,12
56	OGI 6	98	LBI 1,8	1B	LDD 1,11	84	XAD 0,4	AD	XAD 2,13
57	OGI 7	A1	LBI 2,1	1C	LDD 1,12	85	XAD 0,5	AE	XAD 2,14
58	OGI 8	A2	LBI 2,2	1D	LDD 1,13	86	XAD 0,6	AF	XAD 2,15
59	OGI 9	A3	LBI 2,3	1E	LDD 1,14	87	XAD 0,7	B0	XAD 3,0
5A	OGI 10	A4	LBI 2,4	1F	LDD 1,15	88	XAD 0,8	B1	XAD 3,1
5B	OGI 11	A5	LBI 2,5	20	LDD 2,0	89	XAD 0,9	B2	XAD 3,2
5C	OGI 12	A6	LBI 2,6	21	LDD 2,1	8A	XAD 0,10	B3	XAD 3,3
5D	OGI 13	A7	LBI 2,7	22	LDD 2,2	8B	XAD 0,11	B4	XAD 3,4
5E	OGI 14	A8	LBI 2,8	23	LDD 2,3	8C	XAD 0,12	B5	XAD 3,5
5F	OGI 15	B1	LBI 3,1	24	LDD 2,4	8D	XAD 0,13	B6	XAD 3,6
60	LEI 0	B2	LBI 3,2	25	LDD 2,5	8E	XAD 0,14	B7	XAD 3,7
61	LEI 1	B3	LBI 3,3	26	LDD 2,6	8F	XAD 0,15	B8	XAD 3,8
62	LEI 2	B4	LBI 3,4	27	LDD 2,7	90	XAD 1,0	B9	XAD 3,9
63	LEI 3	B5	LBI 3,5	28	LDD 2,8	91	XAD 1,1	BA	XAD 3,10
64	LEI 4	B6	LBI 3,6	29	LDD 2,9	92	XAD 1,2	BB	XAD 3,11
65	LEI 5	B7	LBI 3,7	2A	LDD 2,10	93	XAD 1,3	BC	XAD 3,12
66	LEI 6	B8	LBI 3,8	2B	LDD 2,11	94	XAD 1,4	BD	XAD 3,13
67	LEI 7	00	LDD 0,0*	2C	LDD 2,12	95	XAD 1,5	BE	XAD 3,14
68	LEI 8	01	LDD 0,1	2D	LDD 2,13	96	XAD 1,6	BF	XAD 3,15
69	LEI 9	02	LDD 0,2	2E	LDD 2,14	97	XAD 1,7		
6A	LEI 10	03	LDD 0,3	2F	LDD 2,15	98	XAD 1,8		
6B	LEI 11	04	LDD 0,4	30	LDD 3,0	99	XAD 1,9		
		05	LDD 0,5						
		06	LDD 0,6						
		07	LDD 0,7						

\*\*\* 00 + xx JSR or JMP to page 0, 1, 10, or 14, word xx (0-3F) 00-3F  
 40 + xx JSR or JMP to page 1, 5, 11, or 15, word xx (0-3F) 40-7F  
 80 + xx JSR or JMP to page 2, 6, 12, or 16, word xx (0-3F) 80-BF  
 C0 + xx JSR or JMP to page 3, 7, 13, or 17, word xx (0-3F) C0-FF

## Appendix D

### Directive Summary

Directive	Function	Page
.ADDR	Address constant generation	4-3
.BYTE	Define byte	4-3
.CHIP	Identification of COP400 device	4-14
.CREF	Start cross reference	4-9
.DO	Begin DO loop	4-10
.ELSE	Conditional assembly directive	4-11
.END	Physical end of source program	4-6
.ENDDO/.ENDM	End DO loop	4-10, 5-2
.ENDDO/.ENDM	End macro definition	4-10, 5-2
.ENDIF	Conditional assembly directive	4-12
.ERROR	Generate error message	4-9
.EXIT	Exit DO loop or macro expansion	4-10, 5-3
=	Assignment	4-4
.FORM	Output listing top-of-form	4-7
.IF	Conditional assembly directive	4-11
.IFC	If character directive	4-11
.INCLD	Include disk file source code	4-5
.LIST	Listing output control	4-8
.LOCAL	Begin local region	4-6
.MACRO	Begin macro definition	5-2
.MLOC	Macro local symbol designation	5-2
.OPT	Define COP400 device options	4-14
.PAGE	Set location counter to page address	4-6
.PRINTX	Send message to CRT screen	4-8
.SET	Assign values to variables	4-4
.SPACE	Space n lines on output listing	4-7
.TITLE	Identification of program	4-7
.WORD	8-bit data generation	4-3
. =	Change program counter	4-3
.XREF	Stop cross reference	4-9

## Appendix E

### Programmer's Checklist

1. Is the source program ended by the .END directive?
2. Is each label ended by a colon (:)?
3. Does each comment start with a semicolon (;)?
4. Does each string constant start and end with a single quote (') or double quotes ("")?
5. Are all external statements listed in EXTRN Directives?
6. Do all hexadecimal numbers start with a number (0-9) and end with H?
7. Is there an .ENDIF for each .IF?
8. Is there at most one .ELSE for each .IF?
9. Is there an .ENDM for each .MACRO, IRP, IRPC and REPT?
10. Are all labels defined once and only once?
11. Is the source program well documented?

# Appendix F

## Positive Powers of Two

$n$	$2^n$	$n$	$2^n$
1	2	51	22517 99813 68524 8
2	4	52	45035 99627 37049 6
3	8	53	90071 99254 74099 2
4	16	54	18014 39850 94819 84
5	32	55	36028 79701 89639 68
6	64	56	72057 59403 79279 36
7	128	57	14411 51880 75855 872
8	256	58	28823 03761 51711 744
9	512	59	57646 07523 03423 488
10	1024	60	11529 21504 60684 6976
11	2048	61	23058 43009 21369 3952
12	4096	62	46116 86018 42738 7904
13	8192	63	92233 72036 85477 5808
14	16384	64	18446 74407 37095 51616
15	32768	65	36893 48814 74191 03232
16	65536	66	73786 97629 48382 06464
17	13107 2	67	14757 39525 89676 41292 8
18	26214 4	68	29514 79051 79352 82585 6
19	52428 8	69	59029 58103 58705 65171 2
20	10485 76	70	11805 91620 71741 13034 24
21	20971 52	71	23611 83241 43482 26068 48
22	41943 04	72	47223 66482 86964 52136 96
23	83886 08	73	94447 32965 73929 04273 92
24	16777 216	74	18889 46593 14785 80854 784
25	33554 432	75	37778 93186 29571 61709 568
26	67108 864	76	75557 86372 59143 23419 136
27	13421 7728	77	15111 57274 51828 64683 8272
28	26843 5456	78	30223 14549 03657 29367 6544
29	53687 0912	79	60446 29098 07314 58735 3088
30	10737 41824	80	12089 25819 61462 91747 06176
31	21474 83648	81	24178 51639 22925 83494 12352
32	42949 67296	82	48357 03278 45851 66988 24704
33	85899 34592	83	96714 06556 91703 33976 49408
34	17179 86918 4	84	19342 81311 38340 66795 29881 6
35	34359 73836 8	85	38685 62622 76681 33590 59763 2
36	68719 47673 6	86	77371 25245 53362 67181 19526 4
37	13743 89534 72	87	15474 25049 10672 53436 23905 28
38	27487 79069 44	88	30948 50098 21345 06872 47810 56
39	54975 58138 88	89	61897 00196 42690 13744 95621 12
40	10995 11627 776	90	12379 40039 28538 02748 99124 224
41	21990 23255 552	91	24758 80078 57076 05497 98248 448
42	43980 46511 104	92	49517 60157 14152 10995 96496 896
43	87960 93022 208	93	99035 20314 28304 21991 92993 792
44	17592 18604 4416	94	19807 04062 85660 84398 38598 7584
45	35184 37208 8832	95	39614 08125 71321 68796 77197 5168
46	70368 74417 7664	96	79228 16251 42643 37593 54395 0336
47	14073 74883 55328	97	15845 63250 28528 67518 70879 00672
48	28147 49767 10656	98	31691 26500 57057 35037 41758 01344
49	56294 99534 21312	99	63382 53001 14114 70074 83516 02688
50	11258 99906 84262 4	100	12676 50600 22822 94014 96703 20537 6
		101	25353 01200 45645 88029 93406 41075 2

# Appendix G

## Negative Powers of Two

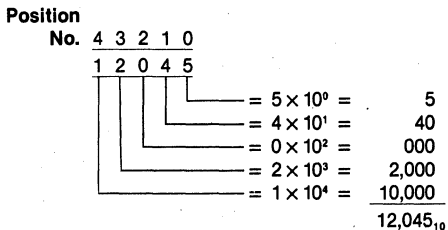
n	2 <sup>-n</sup>									
0	1.0									
1	0.5									
2	0.25									
3	0.125									
4	0.0625									
5	0.03125									
6	0.015625									
7	0.0078125									
8	0.00390625									
9	0.001953125									
10	0.0009765625									
11	0.00048828125									
12	0.000244140625									
13	0.0001220703125									
14	0.00006103515625									
15	0.000030517578125									
16	0.0000152587890625									
17	0.00000762939453125									
18	0.000003814697265625									
19	0.0000019073486328125									
20	0.00000095367431640625									
21	0.000000476837158203125									
22	0.0000002384185791015625									
23	0.00000011920928955078125									
24	0.000000059604644775390625									
25	0.0000000298023223876953125									
26	0.00000001490116119384765625									
27	0.000000007450580596923828125									
28	0.0000000037252902984619140625									
29	0.00000000186264514923095703125									
30	0.000000000931322574615478515625									
31	0.0000000004656612873077392578125									
32	0.00000000023283064365386962890625									
33	0.000000000116415321826934814453125									
34	0.0000000000582076609134674072265625									
35	0.00000000002910383045673370361328125									
36	0.000000000014551915228366851806640625									
37	0.0000000000072759576141834259033203125									
38	0.00000000000363797880709171295166015625									
39	0.000000000001818989403545856475830078125									
40	0.0000000000009094947017729282379150390625									
41	0.00000000000045474735088646411895751953125									
42	0.000000000000227373675443232059478759765625									
43	0.0000000000001136868377216160297393798828125									
44	0.00000000000005684341886080801486968994140625									
45	0.0000000000000284217094304040074348449703125									
46	0.0000000000000142108547152020037174224853515625									
47	0.00000000000000710542735760100185871124267578125									
48	0.000000000000003552713678800500929355621337890625									
49	0.000000000000001776356839400250464678106689453125									
50	0.00000000000000088817841970012523233890533447265625									



# Appendix H

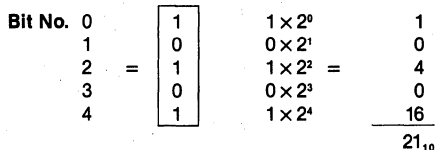
## The Hexadecimal Number System

We have been taught from childhood to recognize and manipulate a number system called decimal or base-10, which uses ten symbols to represent values or numbers. These symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. Combinations of these form other numbers, and each number or digit position is assigned a value equal to its position in the number sequence. For example, the number 12,045:



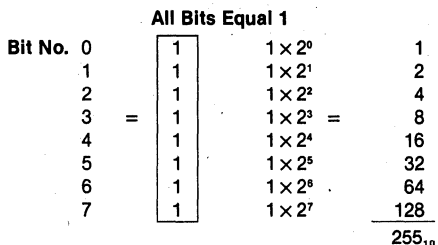
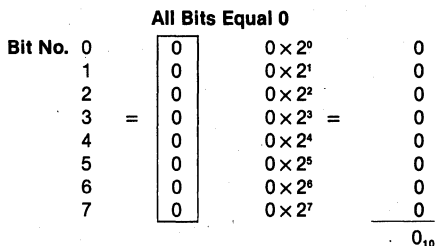
10 is the base value of the number system, and 0, 1, 2, 3, 4 are the positions of weighted values.

Most computers use a base-2 number system in which zeros and ones are the only symbols used to represent any number. The least significant bit would have a value of  $2^0$ , the next bit would be  $2^1$ , then  $2^2$ , etc. Let's use a group of five bits and assign bit 0 as the least significant bit.



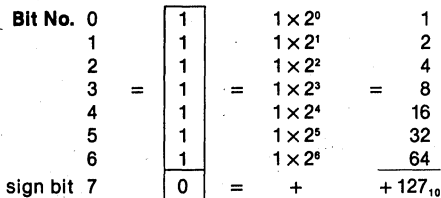
21 is the sum of the values of the bit positions.

It can also be seen that by using larger groups of bits, larger numbers may be represented. An 8-bit computer, which can handle 8-bit positions in parallel, can represent numbers from 0 to 255<sub>10</sub>.



A computer that has 16-bit positions may represent numbers with values from zero to 65,535.

Another consideration in computers is the representation of both positive and negative values. In the "sign magnitude" system, this may be accomplished by assigning one of the bits in a group as a plus/minus indicator. The normal method is to assign the most significant bit position to this task. If it is a logic zero, then the value is positive; if it is a logic one, then the value is negative. Assuming a group of eight bits maximum, and using the eighth position as the sign, we may represent the following numbers:

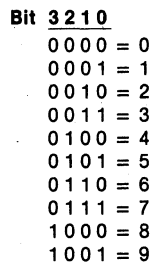


If bit 7 is equal to a 1, then the above number would be negative: -127. Note that by using the most significant bit for the sign, the maximum number that may be represented is only  $\pm 127$ . In a 16-bit computer, this number would be  $\pm 32,767$ .

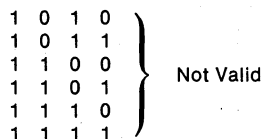
Because it is difficult for us to convert visually many one and zeros to their represented value, other methods of representing numbers have been implemented.

### BCD or Binary Coded Decimal

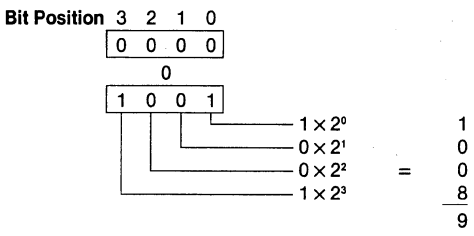
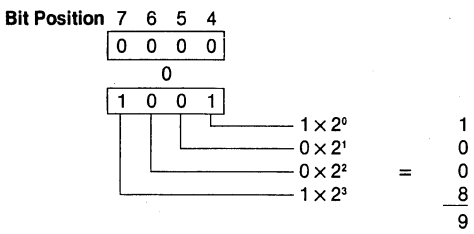
BCD uses groups of four binary bits of positions, and only uses those combinations that add up to 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9. For example:



The other binary combinations possible in the 4-bit positions are not allowed in the BCD method:

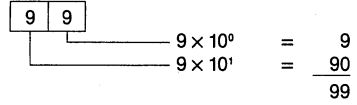


In an 8-bit computer, the decimal numbers 00 through 99 may be represented:



Note that the binary weighting system repeats for each 4-bit group.

This is then compensated for by applying the decimal (base-10) rules to the converted numbers:



(By having to weigh only up to four binary bits, you quickly become efficient at converting binary numbers to decimal form and decimal numbers to binary form.)

The maximum numbers that can be represented in an 8-bit machine is then only 99<sub>10</sub> in decimal versus 255<sub>10</sub> in binary.

As you can see, the efficiency of a computer is restricted because of the illegal combination in each 4-bit group. Another representation of binary numbers allows for all combinations of the four-bit groups. This system is called hexadecimal representation.

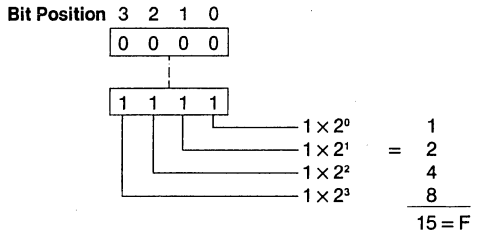
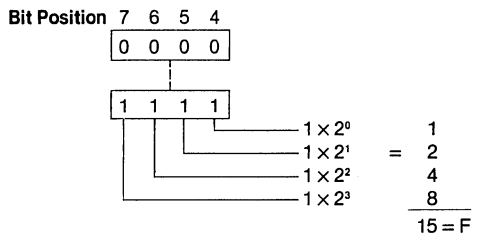
**Hexadecimal (Hex) Notation**

Hex uses a number system of base 16, and allows for all combinations of the 4-bit binary groups, as follows:

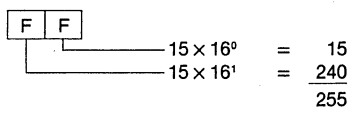
Bit Position	3	2	1	0	Binary	Hex Symbol
	0	0	0	0	0	0
	0	0	0	1	1	1
	0	0	1	0	2	2
	0	0	1	1	3	3
	0	1	0	0	4	4
	0	1	0	1	5	5
	0	1	1	0	6	6
	0	1	1	1	7	7
	1	0	0	0	8	8
	1	0	0	1	9	9
	1	0	1	0	10	A
	1	0	1	1	11	B
	1	1	0	0	12	C
	1	1	0	1	13	D
	1	1	1	0	14	E
	1	1	1	1	15	F

The notations A through F are used to allow for a single-character representation of the four-bit group without duplication.

With hex we can now represent all 16 combinations of binary weights possible in a group of 4-bit positions. An 8-bit computer can then represent the numbers 00 through FF, which is equivalent to binary 0 through 255:



Applying the same rules as for decimal, but using the base 16 instead of base 10:



Thus, binary numbers, no matter what the number of position, can easily be converted simply by dividing them into groups of four bits. For example, in a 16 bit computer:

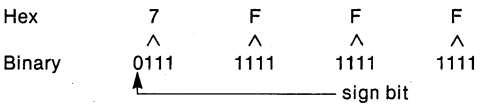
Hex	F	E	9	A
	^	^	^	^
Binary	1111	1110	1001	1010
	v	v	v	v
Hex	F	E	9	A

Further, the use of hex symbols as an equivalent for four binary bits requires fewer printed symbols, and most computer documentation today uses the hexadecimal code representation.

**Positive and Negative Numbers**

In hex or in binary, the method of representing positive and negative numbers is the same. The most significant bit of the most significant group is set to a zero for a positive number or a one for a negative number.

If there are four groups of four bits each, as in a 16-bit computer, we could have:



This number is equivalent to +32,767.

# Appendix I

## Hexadecimal and Decimal Integer Conversion

8		7		6		5		4		3		2		1	
Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	268 435 456	1	16 777 216	1	1 048 576	1	65 536	1	4 096	1	256	1	16	1	1
2	536 870 912	2	33 554 432	2	2 097 152	2	131 072	2	8 192	2	512	2	32	2	2
3	805 306 368	3	50 331 648	3	3 145 728	3	196 608	3	12 288	3	768	3	48	3	3
4	1 073 741 824	4	67 108 864	4	4 194 304	4	262 144	4	16 384	4	1 024	4	64	4	4
5	1 342 177 280	5	83 886 080	5	5 242 880	5	327 680	5	20 480	5	1 208	5	80	5	5
6	1 610 612 736	6	100 663 296	6	6 291 456	6	393 216	6	24 576	6	1 536	6	96	6	6
7	1 879 048 192	7	117 440 512	7	7 340 032	7	458 752	7	28 672	7	1 792	7	112	7	7
8	2 147 483 648	8	134 217 728	8	8 388 608	8	524 288	8	32 768	8	2 048	8	128	8	8
9	2 415 919 104	9	150 994 944	9	9 437 184	9	589 824	9	36 864	9	2 304	9	144	9	9
A	2 684 354 560	A	167 722 160	A	10 485 760	A	655 360	A	40 960	A	2 560	A	160	A	10
B	2 952 790 016	B	184 549 376	B	11 534 336	B	720 896	B	45 056	B	2 816	B	176	B	11
C	3 221 225 472	C	201 326 592	C	12 582 912	C	786 432	C	49 152	C	3 072	C	192	C	12
D	3 489 660 928	D	218 103 808	D	13 631 488	D	851 968	D	53 248	D	3 328	D	208	D	13
E	3 758 096 384	E	234 881 024	E	14 680 064	E	917 504	E	57 344	E	3 584	E	224	E	14
F	4 026 531 840	F	251 658 240	F	15 728 640	F	983 040	F	61 440	F	3 840	F	240	F	15

### To Convert Hexadecimal to Decimal

1. Locate the column of decimal numbers corresponding to the left-most digit or letter of the hexadecimal; select from this column and record the number that corresponds to the position of the hexadecimal digit or letter.
2. Repeat step 1 for the next (second from the left) position.
3. Repeat step 1 for the units (third from the left) position
4. Add the numbers selected from the table to form the decimal number.

### To Convert Decimal to Hexadecimal

1. (a) Select from the table the highest decimal number that is equal to or less than the number to be converted.  
 (b) Record the hexadecimal of the column containing the selected number.  
 (c) Subtract the selected decimal from the number to be converted.
2. Using the remainder from step 1(c) repeat all of step 1 to develop the second position of the hexadecimal (and a remainder).
3. Using the remainder from step 2 repeat all of step 1 to develop the units position of the hexadecimal.
4. Combine terms to form the hexadecimal number.

To convert integer numbers greater than the capacity of table, use the techniques below:

#### Hexadecimal to Decimal

Successive cumulative multiplication from left to right, addition units position.

Example:  $D34_{16} = 3380_{10}$

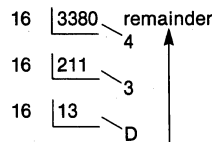
$$\begin{array}{r}
 D = 13 \\
 \times 16 \\
 \hline
 208 \\
 3 = +3 \\
 \hline
 211 \\
 \times 16 \\
 \hline
 3376 \\
 4 = +4 \\
 \hline
 3380
 \end{array}$$

<b>Example</b>	
Conversion of Hexadecimal Value D34	
D	3328
3	48
4	4
Decimal	3380

#### Decimal to Hexadecimal

Divide and collect the remainder in reverse order.

Example:  $3380_{10} = D34_{16}$



<b>Example</b>	
Conversion of Decimal Value 3380	
D	- 3328
	52
3	- 48
	4
4	- 4
Hexadecimal	D34

# Appendix J

## Hexadecimal and Decimal Fraction Conversion

1		2		3		4					
Hex	Decimal	Hex	Decimal	Hex	Decimal	Hex	Decimal				
.0	.000	.00	.0000 0000	.000	.0000 0000 0000	.0000	.0000 0000 0000 0000				
.1	.0625	.01	.0039 0625	.001	.0002 4414 0625	.0001	.0000 1525 8789 0625				
.2	.1250	.02	.0078 1250	.002	.0004 8828 1250	.0002	.0000 3051 7578 1250				
.3	.1875	.03	.0117 1875	.003	.0007 3242 1875	.0003	.0000 4577 6367 1875				
.4	.2500	.04	.0156 2500	.004	.0009 7656 2500	.0004	.0000 6103 5156 2500				
.5	.3125	.05	.0195 3125	.005	.0012 2070 3125	.0005	.0000 7629 3945 3125				
.6	.3750	.06	.0234 3750	.006	.0014 6484 3750	.0006	.0000 9155 2734 3750				
.7	.4375	.07	.0273 4375	.007	.0017 0898 4375	.0007	.0001 0681 1523 4375				
.8	.5000	.08	.0312 5000	.008	.0019 5312 5000	.0008	.0001 2207 0312 5000				
.9	.5625	.09	.0351 5625	.009	.0021 9726 5625	.0009	.0001 3732 9101 5625				
.A	.6250	.0A	.0390 6250	.00A	.0024 4140 6250	.000A	.0001 5258 7890 6250				
.B	.6875	.0B	.0429 6875	.00B	.0026 8554 6875	.000B	.0001 6784 6679 6875				
.C	.7500	.0C	.0468 7500	.00C	.0029 2968 7500	.000C	.0001 8310 5468 7500				
.D	.8125	.0D	.0507 8125	.00D	.0031 7382 8125	.000D	.0001 9836 4257 8125				
.E	.8750	.0E	.0546 8750	.00E	.0034 1796 8750	.000E	.0002 1362 3046 8750				
.F	.9375	.0F	.0585 9375	.00F	.0036 6210 9375	.000F	.0002 2888 1835 9375				
1		2		3		4					

### To Convert .ABC Hexadecimal to Decimal

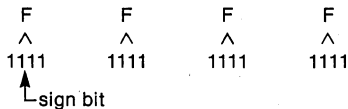
Find .A in position 1 .6250

Find .0B in position 2 .0429 6875

Find .00C in position 3 .0029 2968

.ABC Hex is equal to .6708 9843 7500

By making the most significant bit a logic 1, the number becomes:



This number is equivalent to -32,767.

The method used to represent a negative hexadecimal number depends on the type of numbering system chosen for binary arithmetic processing. Most digital computers use either the "sign magnitude" system or

the two's-complement system. In the sign magnitude system, a negative value is formed by setting a sign bit—the most significant bit of the most significant group of bits—to one, and the remaining bits to the desired absolute value. Thus, -32,767 is represented as 1111 1111 1111 1111.

Conversely, if the most significant bit is a zero, the number is positive; +32,767 is represented as 0111 1111 1111 1111.

In the two's-complement system—the method used in the STARPLEX™ System—positive numbers are represented as in the sign magnitude system (sign bit is a logic zero); but negative numbers are represented by the two's-complement of the absolute value of the number. Thus, -32,767 becomes, in the two's-complement system, 1000 0000 0000 0001.



# Appendix K

## Negative Hexadecimal Numbers

The 8080 microprocessor maintains negative numbers in twos-complement form. To convert a number in hexadecimal notation to its twos-complement equivalent, subtract the number from hexadecimal  $2^n$ , where "n" is the number of binary bits in the computer word. For a 16-bit word, "n" is 16, and  $2^n$  is 1 0000 0000 0000 0000 (binary) or 1 0000 (hex).

Thus, the negative of  $1245_{16}$  is:

$$\begin{array}{r} 10000 \\ - 1245 \\ \hline \text{EDBB} \end{array}$$

A hexadecimal number will be negative in the 8080 CPU if the left-most digit is 8, 9, A, B, C, D, E, or F (because all of these groupings start with a one). Thus, the twos-complement of hex FACE is:

$$\begin{array}{r} 10000 \\ - \text{FACE} \\ \hline + 0532 \end{array}$$

Perhaps an easier way to find the twos-complement of a hexadecimal number is first to take the ones-complement of the number; the ones-complement plus one is the twos-complement. The ones-complement of a number is its inverted form; simply exchange its ones for zeros, and its zeros for ones. Thus,

hexadecimal	binary equivalent	ones-complement
FACE	→ 1111 1010 1100 1110	→ 0000 0101 0011 0001

	ones-complement + 1
	0000 0101 0011 0001
	+ 1
	0000 0101 0011 0010

Hex twos-complement of FACE →

0 5 3 2

# Appendix L

## Program Listing Format

### 1. Listing without cross reference table.

PRINT POSITION

```

0          1          2          3          4          5          6          7          8
1234567890123456789012345678901234567890123456789012345678901234567890
  
```

```

NSC STARPLEX 8080 MACRO-ASSEMBLER VERSION 1.0                                PAGE NNNN
PROGRAM-----FILEID.EXT
PROGRAM SUBTITLE-----
  
```

```

1 22223 44444444          555 ..... 555
  
```

USER SOURCE LINE EXACTLY AS IN SOURCE FILE  
 MAXIMUM SIZE 56 BYTES, LONGER LINES TRUNCATED.

GENERATED OBJECT CODE

ADDRESS TYPE

LOCATION COUNTER

ERROR CODE (SEE APPENDIX N)

### 2. Listing with cross reference table.

PRINT POSITION

```

0          1          2          3          4          5          6          7          8
1234567890123456789012345678901234567890123456789012345678901234567890
  
```

```

NSC STARPLEX 8080 MACRO-ASSEMBLER VERSION 1.0                                PAGE NNNN
PROGRAM-----FILEID.EXT
PROGRAM SUBTITLE-----
  
```

```

00000 1 22223 44444444          555 ..... 555
  
```

USER SOURCE LINE EXACTLY AS IN SOURCE FILE  
 MAXIMUM SIZE 48 BYTES, LONGER LINES TRUNCATED.

GENERATED OBJECT CODE

ADDRESS TYPE

LOCATION COUNTER

ERROR CODE (SEE APPENDIX N)

LINE NUMBER



# Appendix N

## Assembler Error Messages

### N.1 ERROR MESSAGES

MISSING ARGUMENT  
ARGUMENT VALUE OUT OF RANGE  
JUMP ADDRESS OUT OF RANGE  
ILLEGAL INSTRUCTION  
SYNTAX ERROR  
EXCESS ARGUMENTS  
UNDEFINED SYMBOL  
DUPLICATE DEFINITION  
PAGE OVERFLOW  
PARAM OVERFLOW (Reference to missing parameter)  
NESTING ERROR  
ILLEGAL DIGIT  
ROM 0 NOT 0  
INSTRUCTION INVALID FOR 410/411  
INSTRUCTION INVALID FOR 410/411/444/445  
INSTRUCTION INVALID FOR 410/411/421/445  
INCLUDE NESTING  
ILLEGAL USE OF HIGH/LOW OPERATOR  
EXPANSION ERROR

### N.2 WARNING MESSAGES

WARNING, END OF BLOCK  
WARNING, PASS 1,2 DISAGREE IN VALUE OF SYMBOL  
CHIP GIVEN ILLEGAL, DEFAULT 420

# **SPM-A15 SPM90/A15 COP400 Emulator**

## **User's Manual**



**Publication No. 420306254-001  
Order No. 420306254-001  
August 1981**

# Preface

This manual describes the COPS™ ISE™ (In-System Emulation) Subsystem, a combination of hardware and software, that gives the user easy access to the registers and RAM memory of the COPS microcontroller for the development of programs and hardware debugging of COPS microcontroller-based systems. The ISE Subsystem allows the user to edit and assemble COPS programs, emulate and test COPS chips, and transmit mask patterns.

The COPS ISE Subsystem is designed for installation in National's STARPLEX™ or STARPLEX II™ Development Systems. It consists of an ISE (In-System Emulation) Board, a COPS Emulator Board, a STARPLEX/Emulator Cable, COPS Emulation Cables, and a diskette containing the system software.

The COPS ISE Subsystem software is compatible with STARPLEX Development System SPX-80/xx (software 440305288-20x Rev. G). It will be compatible with STARPLEX II Development System SPX-90/xx by the third quarter 1981.

The following manuals provide further information on COPS and the STARPLEX Development System:

- STARPLEX COPS Cross-Assembler Software User's Manual  
Publication No. 420306253-001
- COP400 In-System Emulator Cards User's Manual  
Publication No. 420306469-001
- COP400 Product Development System User's Manual  
Publication No. 420305528-001
- COPS CHIP User's Manual  
Publication No. 420305785-001
- STARPLEX System Software Reference Manual  
Publication No. 420305788-001
- STARPLEX System Hardware Reference Manual  
Publication No. 420305789-001
- STARPLEX II System Software Reference Manual  
Publication No. 420306383-001
- STARPLEX II System Hardware Reference Manual  
Publication No. 420306465-001

The material presented in this manual is for information purposes only as specifications for both the COPS ISE Subsystem and the STARPLEX System are subject to change without notice.

# SPM A-15

## Table of Contents

Section	Description	Page
<b>Chapter 1. Introduction and Overview</b>		
1.1	General Description .....	8-243
1.2	Emulation and Debugging .....	8-243
1.3	COPS ISE .....	8-244
1.4	Basic System Configuration .....	8-244
1.5	COPS Software .....	8-245
<b>Chapter 2. Specifications</b>		
2.1	General .....	8-246
2.2	COPS ISE Board .....	8-246
2.3	Emulator Board .....	8-246
2.4	Cable Assemblies .....	8-246
<b>Chapter 3. System Installation and Setup</b>		
3.1	General .....	8-247
3.2	Unpacking and Inspection .....	8-247
3.3	Installation .....	8-247
3.4	Jumpers and User Options .....	8-248
3.5	Installation Checkout .....	8-248
<b>Chapter 4. COPS Monitor and Debugger (COPMON)</b>		
4.1	COPMON (COP Monitor) .....	8-249
4.2	Console Operation .....	8-249
4.3	COPMON Console Commands .....	8-250
<b>Chapter 5. Mask Transmittal Program (MASKTR)</b>		
5.1	MASKTR .....	8-261
5.2	MASKTR Console Commands .....	8-261
5.3	MASKTR Example .....	8-263
<b>Illustrations</b>		
<b>Figure</b>		<b>Page</b>
2-1	COPS ISE Board .....	8-246
3-1	Installation of the SPM-A15 and an Emulator Board .....	8-248
<b>Tables</b>		
<b>Table</b>		<b>Page</b>
4-1	Valid Chip Numbers .....	8-257
4-2	Summary of COPMON Console Commands .....	8-257
4-3	Operand Syntax .....	8-258
4-4	GO Operation Summary .....	8-259
4-5	Keyword Abbreviations .....	8-260
5-1	Summary of MASKTR Console Commands .....	8-266

# Introduction and Overview

## 1.1 General Description

The COPSTM ISE™ Subsystem is designed for installation in National's STARPLEX™ or STARPLEX II™ Development Systems. It extends the use of these systems to the development of programs and hardware debugging of COPS microcontroller-based systems. As such, it provides the user with four major capabilities:

1. Editing COPS programs. Creates new programs and/or modifies existing programs.
2. Assembling programs. The output of the assembler is object code that can be executed by a COPS microcontroller-based system.
3. Emulation. Allows the user to execute COPS object code. The unit can execute this code either through a prototype system under test or internally without the prototype. This allows much of the target program to be developed while the prototype is being designed and constructed. Later, the COPS ISE system will execute the program in the actual prototype, enabling the debugging of both hardware and software.
4. Transmitting mask patterns. Because COPS programs generally are meant to be encoded into a ROM pattern on the COPS chip itself, some method of transmitting the program information to National is necessary so that the correct semiconductor masks can be fabricated with the appropriate ROM pattern. If this were done with paper and pencil, there would be a large potential for errors. The COPS ISE Subsystem solves this program by creating diskettes that contain the ROM pattern data in a format that can be read directly at National.

The first two functions above are covered in other manuals. New programs are edited and created by using the STARPLEX Editor. Once a program has been created with the editor, it can be assembled using the COPS Assembler. The operation of this assembler and the instruction mnemonics recognized by it are covered in detail in the COPS Assembly Language Manual listed in the preface of this manual.

Functions 3 through 4, that is, the emulation, transmission of mask patterns, are covered in this manual.

## 1.2 Emulation and Debugging

Because the process of debugging programs and prototypes is so important, we will pause and consider the problems a user faces in bringing up a prototype system for the first time. Then, with this process in perspective, we can consider the operation of the COPS ISE Subsystem and how it solves those problems.

Assume that the user has created a COPS program, assembled it, and that program, in machine coded form, now resides on a floppy diskette. At the same time, he has constructed a prototype system to run the program. His immediate problem is to somehow load the program on the diskette into the prototype and then to determine if the prototype correctly executes that program.

The COPS microcontroller family consists of a series of single-chip microcontrollers, each containing CPU, RAM, I/O, and some specialized functions such as interrupts and a time/counter. In addition, most of the devices in the family contain a mask-programmable ROM. That ROM is intended to store the program that the COPS chip will execute. Because the program is in ROM, it cannot be easily changed or modified by the user during the development phase of the project. The problem of loading the program into the prototype is complicated because the COPS chip stores the program as a ROM pattern on the COPS chip itself. So some means must be found of storing the program, not in ROM, but in RAM where it can be easily changed as the debugging process proceeds. This implies that some means of having the COPS do instruction fetches from the RAM rather than the on-chip ROM has been found; we will discuss this in more detail in a moment. For now, assume that the system contains some RAM, referred to in this manual as "shared memory," and that the program can be stored in this shared memory RAM.

Of course, just having some appropriately placed RAM is not sufficient. Some means of transferring or loading the program from the diskette is also needed. In fact, as we go through this discussion, other functions will be developed, which must be controlled by the user from the system console. The loading and translation process and the other required functions are performed by COPMON, the COPS monitor program. After the program has been loaded, the COPS chip must be able to execute it, and COPMON must be able to start the COPS chip, stop it, single-step it, and so on.

The COPS microcontroller is also part of the emulation system. It performs instruction fetches from the program stored in the shared memory RAM, executes them, and performs the appropriate actions. The microcontroller in the emulation system is connected via a cable to the microcontroller socket in the prototype. Since the microcontroller is electrically connected to the prototype through a cable, it exercises all of the I/O pins on that socket, just as the actual microcontroller would. As a result, the prototype should function in the same way as the production unit with the ROM-masked COPS chip—that is, if the program is correct.



Of course, the program will not be correct on the first pass, which brings the user to a quandary: What is wrong with the program and how can it be tested? Probably some of the functions of the program will appear to be operational and others will not. In extreme cases, a programming error early on will make the entire prototype nonfunctional, so the user cannot diagnose the problem by reviewing the symptoms. However, with the monitor program, the user can insert breakpoints in the program. Breakpoints interrupt the normal flow of the program and pass control back to the monitor. The monitor can be used to examine and modify the registers in the COPS™ chip. By placing the breakpoints at strategic points in the program, the user can determine if those points are being reached in the program flow; and if they are, do the registers contain the values expected? Since the user can modify the contents of the registers before resuming program execution at that point, he can perform software experiments. For instance, if he had expected one of the registers to contain value A and instead it contains value B, does replacing B with A then make the function work? If it does, only the question of how B got into the register remains. Once the ability to construct these experiments has been mastered, the operator can quickly locate the problem in most programs.

The simple ability to set breakpoints and examine and modify the registers allows most programs to be successfully debugged. However, the added ability to single-step through the program can make debugging even easier. Assume that one of the program modules was not operating correctly, and the appropriate use of breakpoints had determined that the module was being entered with the correct values in the registers, but somehow they were being changed erroneously during the execution of that software or hardware. Single-stepping is an extension of the breakpoint function, but it is much faster to use once the problem has been localized to a small section of the program.

### 1.3 COPS ISE™

The COPS ISE Subsystem neatly solves all of the previously discussed problems. It is a combination of software and hardware, integrated to give the user easy access to the registers and RAM memory of the COPS microcontroller. The software includes a monitor program that can be used to load machine coded programs from diskettes into the shared memory RAM on the ISE Subsystem. A ROMless microcontroller chip executes the code contained in the shared memory just as though it were contained in the ROM on the chip. The COPS chip is connected to the prototype system through an emulation cable plugged into the socket of the prototype board that would normally contain the COPS chip. Although the instruction fetches are from shared memory RAM on the ISE Subsystem, all other external functions of the microcontroller occur through the emulation cable. That means that all of the I/O activity on the prototype board will occur just as though the COPS chip was resident in the socket on the prototype board. Logic in the ISE Subsystem monitors the addresses on the bus connecting the COPS ROMless microcontroller chip to

the program memory. When this logic detects a pre-programmed condition (a breakpoint address) the execution of the program is interrupted and the monitor program is entered. Now the operator can examine the registers of the controller. This is possible, because encountering a breakpoint causes the internal registers of the COPS chip to be saved in RAM dedicated for this purpose. Examining and modifying registers actually accesses these RAM locations. Then, when the user instructs the program execution to resume, the monitor reloads the registers with the contents of the appropriate image locations in RAM and begins executing the user program. In addition, single-step circuits allow the user to single-step through portions of the user program.

Another program provides trace memory for user program execution. This means that the operator can set a condition for trace, execute his program, and if the trace condition is met, 256 instruction fetches will be stored in the trace memory where they can be examined by the operator. In actual practice, the trace memory is constantly being loaded whenever the trace flag is enabled. When the condition for trace is set, the number of cycles prior to trace point are also set. Then, when the trace point occurs, the trace memory continues to load instruction fetches for the appropriate number of cycles AFTER the trace point. This allows the operator to examine the "history" of the program execution both before and after the trace point.

### 1.4 Basic System Configuration

The basic COPS ISE Subsystem consists of an ISE Board, a COPS Emulator Board, a STARPLEX™/Emulator Cable, COPS Emulation Cables, and a diskette containing COPMON and MASKTR. The ISE Board is in the BLC configuration and is designed for installation in the STARPLEX System. As such, it appears as a set of I/O ports in the STARPLEX I/O address map. All communication with the COPS ISE Subsystem is via these I/O ports. The COPMON program is loaded and executed on the STARPLEX as any other program. It provides a set of console commands for interfacing to the ISE Board. The user need only become familiar with these commands; the actual interfacing is done by COPMON. The details of the I/O addressing and data interchange are totally transparent to the operator.

The ISE Board contains 4k bytes of shared RAM memory, the trace memory, the trigger logic for breakpoints and single-steps, and all of the necessary control logic. The STARPLEX/Emulator Cable connects the ISE Board in the STARPLEX System to the Emulator Board which is placed in close physical proximity to the prototype under development. The Emulator Board contains the ROMless COPS chip and a set of buffers for buffering the STARPLEX Emulator Cable signals. Signals through the COPS Emulation Cable are *not* buffered. The COPS Emulation Cables connect the Emulator Board to the COPS socket on the user's prototype board.

Ordering information for the various components of the COPST<sup>™</sup> ISE<sup>™</sup> Subsystem is as follows:

- SPM-A15, consists of the ISE board, STARPLEX<sup>™</sup>/Emulator Cable, COPMON/MASKTR diskette and user documentation.
- COP400-E02, E04L, consists of the Emulator Board (E02 or E04L), the Emulation Cables, and user documentation.
- COP400-E24, consists of the Emulator Board, Emulation Cables, and user documentation. Used for emulation of COP440, COP441, COP442, COP2440, COP2441, and COP2442.
- SPM90/A15, consists of the ISE board, the STARPLEX Emulator Cable, the COPMON/MASKTR diskette and user documentation for use with a STARPLEX II Development System.

## 1.5 COPS Software

This section covers the various COPS programs that are included on the National supplied COPS STARPLEX System Software Diskette.

### 1.5.1 COPMON (Monitor)

The basic interface to the ISE Board is via the COPMON program. It includes all of the console commands used to load programs from diskette into the shared memory on the ISE board, set breakpoint and trace conditions, execute the program, examine and modify registers, change memory locations, single-

step the program, examine the trace memory, etc. Virtually all of the operator interface with the ISE system during program debug and hardware checkout will be done through the COPMON program. The COPMON program is covered in detail in Chapter 4.

### 1.5.2 Assembler

The COPS Assembler is shipped with every ISE unit and is contained on its own diskette. It assembles COPS programs written with the STARPLEX Editor and stores them as object code load modules on the system diskettes. There they are accessible to the COPMON program which loads them into the shared memory on the ISE Board and executes them through the Emulator Board. The COPS Assembler is covered in a separate publication, the COPS Assembler Manual listed in the preface of this manual.

### 1.5.3 MASKTR

The second program on the COPS STARPLEX System Software diskette is the MASKTR program. It accepts object code load modules prepared by the assembler as input files and translates them into a transmittal file which is stored on another diskette. This transmittal file is in a format that can be used by National to prepare the masks required to manufacture ROM-based COPS chips. MASKTR is only used at the completion of the debugging process. It is discussed in detail in Chapter 5.

# Specifications

## 2.1 General

This chapter details the characteristics and specifications of the COPS™ ISE™ and Emulator Board.

## 2.2 COPS ISE Board

### 2.2.1 Physical Specifications

The COPS ISE Board is a 12 × 6.75-inch BLC-format printed circuit board intended for installation in the STARPLEX™ mainframe card cage. See *Figure 2-1*.

### 2.2.2 Environmental Specifications

Normal precautions must be taken to avoid temperature and humidity extremes. Since the ISE Board will be installed in the STARPLEX mainframe card cage, the normal precautions applicable to the system will ensure that the conditions necessary for satisfactory operation of the ISE Board will have been met. This means that the operating temperatures for the system should not exceed 10°C to 32°C ambient temperature range and the relative humidity should not exceed 90% noncondensing.

### 2.2.3 Power Specifications

The ISE Board receives its power from the STARPLEX System. It requires only +5 volts at a maximum current of 3.5 amps.

## 2.3 Emulator Board

The COPS Emulator Board is located outside of the STARPLEX System and in close physical proximity to the prototype system under development. While it is normally intended for use in conjunction with the ISE Board, it is possible to use the Emulator Board in a stand-alone mode connected to the user's prototype through the COPS Emulation Cable. In this mode, the ROMless COPS chip does its instruction fetches from EPROM sockets provided on the Emulator Board. For more details on the specifications and use of the

Emulator Boards, refer to the users' manuals listed in the preface of this manual.

## 2.3.1 Physical Specifications

At one end of the Emulator Board is a 50-pin edge connector. It receives the STARPLEX/Emulator Cable that plugs into the ISE Board connectors J1 and J2. The pinout assignment of this connector is listed in Appendix A of this manual.

## 2.3.2 Environmental Specifications

Normal precautions must be taken to avoid temperature extremes. Guaranteed operation can be expected if the temperature is maintained between 0°C and 55°C. Relative humidity should not exceed 90% noncondensing.

## 2.3.3 Power Specifications

The Emulator Board receives its power from the ISE Board through the STARPLEX Emulator cable. Although the Emulator Board has a separate set of connections for +5V and -12V, these are for powering the board in the event that it is used independently without the ISE Board. **THESE CONNECTIONS ARE NOT INTENDED FOR ACCESSING THE STARPLEX POWER SUPPLIES!** The Emulator Board requires only 500mA at +5V. However, if it is used in the stand-alone mode with MM5204 EPROMs, an additional -12V supply is needed to power the PROMs on the Emulator Board.

## 2.4 Cable Assemblies

When installed in the STARPLEX Development System as directed in Chapter 3, the overall reach of the various cable assemblies are approximately four feet from the STARPLEX Development System to the user's application system.

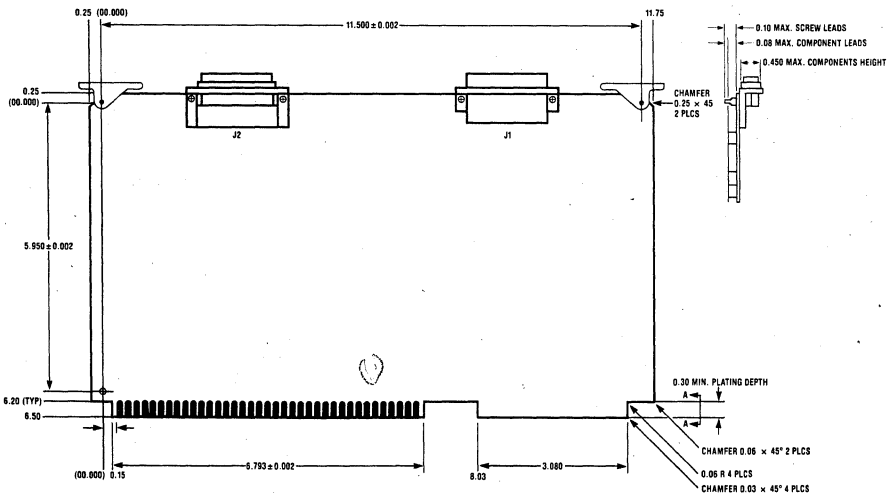


Figure 2-1. COPS ISE™ Board

# System Installation and Setup

## 3.1 General

This chapter describes the initial installation of the ISE™ Board in the STARPLEX™ or STARPLEX II™ Development Systems as well as the connection procedures to the COPS™ Emulator board. Power-up and software loading is also described.

## 3.2 Unpacking and Inspection

All of the COPS ISE Subsystem modules are individually tested at the factory during manufacture. However, improper handling practices during shipment may cause damage to the equipment, which if undetected, can create unnecessary problems in checking out the unit after it is installed.

Before accepting the equipment from a carrier, inspect all shipping containers for evidence of external damage. Any indication of external damage must be noted by both the recipient and carrier. Carefully unpack the equipment, and before discarding the packing material, check to determine that everything is intact. All packages listed on the shipment billing should be accounted for.

Carefully unwrap and inspect all of the modules and cables for evidence of shipping damage. Look for scratched PC boards, bends or creases in the floppy diskettes, sharp bends in the cables, etc. If such evidence is present, stop unpacking as soon as the damage is discovered, notify the carrier and arrange to have the shipment inspected by the carrier's agent or authorized representative immediately. All claims for damage should be filed promptly with the transportation company involved.

Any returns to the factory must be packed in either the original container or a substitute container of equal strength and durability. A description of the equipment defect, the nature of its cause, and the name and address of the sender, should accompany each return shipment.

Returned equipment is to be sent to the following address:

Microcomputer Service Center  
675 Almanor Avenue  
Sunnyvale, CA 94086  
Attn: Microcomputer Service Manager  
Mail Stop 15205  
Telephone: (408) 721-5883

Correspondence should be sent to the following address:

*Domestic Contact*  
National Semiconductor Corp.  
2900 Semiconductor Drive  
Santa Clara, CA 95051  
Attn: Microcomputer Service Mgr.  
Mail Stop: 15205  
Telephone: (408) 721-6279

### *European Contact*

National Semiconductor GmbH  
Industriestrasse #10  
D8080 Fuerstenfeldbruck  
West Germany  
Telephone (08141) 1371  
Telex: 05-27649

In other countries, contact your local National Semiconductor Sales Office or authorized representative.

## 3.3 Installation

Installation of the SPM-A15 (or SPM90/A15) and any of the various emulator boards is illustrated in *Figure 3-1* and discussed in the following text.

### 3.3.1 ISE Board

The ISE Board is designed for installation in the expander card cage contained within the STARPLEX System. This cage is located at the left side of the STARPLEX Base Module as viewed from the operator position. It is accessed by opening the card cage access door located on the left side of the base module. Install the ISE Board according to the following procedure:

1. Turn the STARPLEX power switch to OFF.
2. Open the card cage access door on the left side of the base module, as viewed from the operator position.
3. Insert the ISE Board into any of the unused slots in the card cage. Make sure the board is correctly positioned in the card guides at the sides of the cage. Then slide the board into the cage until the edge connector fingers reach the edge connector at the back of the cage. The component side of the board should be facing up. Gripping both ends of the cage with your fingers, assert pressure on the ends of the board with your thumbs to push the board home into the socket. As soon as the board has initially entered the edge connector, it may be completely inserted by pressing against the toggle handles at each end of the board.
4. Now connect the STARPLEX/Emulator cable to the ISE Board in the STARPLEX card cage. The cable is keyed so that it cannot be connected incorrectly. The STARPLEX ISE end of the cable is terminated with two separate connectors: a 25-pin D-type MALE connector and a second 25-pin D-type FEMALE connector. Plug the MALE connector into the corresponding FEMALE connector on the ISE Board. Then plug the FEMALE connector into the corresponding MALE connector on the ISE Board.
5. Close the card cage access door, ensuring that the STARPLEX/Emulator cable is not crimped between the door and the enclosure.

The installation of the ISE Board is now complete.

### 3.3.2 COPS™ Emulator Board

The COPS Emulator Board is designed to operate as a free-standing board. Four 0.5-inch nylon stand-offs on the bottom of the board raise the board off the surface on which it is resting. Although there are no mounting procedures to be followed when using the Emulator Board, care should be taken to be sure the working surface is free of metal tools, pieces of wire, or other metallic objects that might cause shorts on the board.

The Emulator Board is connected electrically to the STARPLEX™ Development System through the STARPLEX/Emulator cable that was connected to the ISE™ Board above. The remaining end of the STARPLEX/Emulator cable is terminated in a single card-edge connector (50-connector) that must be connected to the edge of the Emulator Board. This cable supplies power to the Emulator Board as well as supplying all control and data transfers between the Emulator Board and the ISE Board.

**BE SURE THAT PIN 1 OF THIS CONNECTOR LINES UP WITH PIN 1 OF THE CABLE.** Damage to the Emulator or the ISE Board will result if this cable is connected incorrectly. Also, do *not* install (or disconnect) this cable while the power is still on, as this also will result in damage to the Emulator or the ISE Board.

All that remains to complete the installation is to connect the COPS Emulation Cable between the user's prototype system and the Emulator Board. A DIP socket on the Emulator Board receives one end of the COPS Emulation Cable. The other end is plugged into the COPS microcontroller socket in the prototype system. This completes the installation of the COPS ISE Subsystem.

This discussion has assumed that the program execution is performed in the user's prototype system. It is possible to run the ISE Emulator Board and COPS Emulation Board without being connected to a prototype system. All portions of the program that do not depend upon data being inputted through the I/O pins will operate correctly. During the early stages of the development project, this allows the user to debug portions of the program without the prototype being

operational. Later, when the prototype becomes available, the program can be executed using the full set of COPS I/O pins.

The Emulator Board has one additional feature not yet mentioned. Four TTL inputs on the board, labeled EXT1-EXT4, can be connected by the user to points in the prototype system. During trace operations, the states of these four inputs will be stored with the other information in trace memory. This allows the user to monitor asynchronous events during program execution. In addition, two of the inputs, EXT1 and EXT2, can be used to initiate trace or breakpoint or time operations. For further installation information on the Emulator Board, refer to the manual provided with that product.

### 3.4 Jumpers and User Options

#### 3.4.1 ISE Board

The ISE Board has only one user configuration. This board has five jumpers, but none are intended to be altered by the user. The standard shipping configuration is as follows:

Jumper	Description
W1	A to B (Future Option)
W2	A to B (I/O Port Page 00)
W3	A to C (I/O Port Address 10)
W4	A to B (Bus Priority Enabled)
W5	Open (-12 Volts to the Emulator)

#### 3.4.2 Emulator Board

The Emulator Board has several user options and jumpers. For detailed information on the use of these options, refer to the manual provided with the Emulator Board.

### 3.5 Installation Checkout

To quickly verify the functional operation of the installed ISE Subsystem, refer to the checkout procedure at the end of the Emulator Board User's manual.

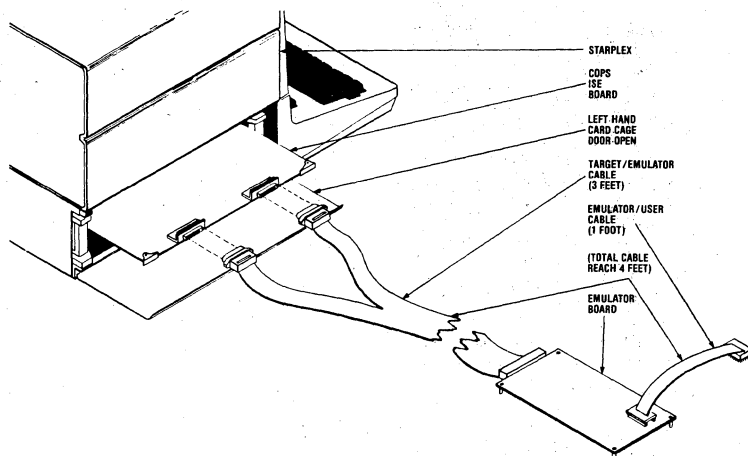


Figure 3-1. Installation of the SPM-A15 and an Emulator Board

# COPS™ Monitor and Debugger (COPMON)

## 4.1 COPMON (COP Monitor)

COPMON is the control program that performs the interface between the STARPLEX™ console and the COPS ISE™ Subsystem. This chapter discusses the format of the COPMON commands and their use in debugging programs and hardware.

COPMON allows the user to interrupt the flow of a COPS program as it is being executed on a prototype system. The interruption is directly caused by one of several events, all under user control. For instance, the interruption may be caused by the COPS chip performing an instruction fetch from a predetermined point in the program called a breakpoint. Once the flow has been interrupted, the COPS registers can be examined and modified. COPMON also allows the user to examine the trace of the program flow for the last 256 instruction cycles, either before or after a specified condition was met. This is called a trace. Possible conditions for a breakpoint or trace may be the program encountering a specified address (address), the next value of the program counter (immediate), or any combination of two external events on the Emulation Board called EXT1 and EXT2.

The TRACE command allows the user to specify the conditions that will initiate the trace and how many steps prior to meeting that condition will be traced. The GO command then arms the trace and executes the program. After a trace has been completed, the operator may examine the trace data with a TYPE command or search for an address in the trace memory with a SEARCH command. By comparing the execution sequence revealed in the trace memory with the expected sequence of instructions, deviations resulting from incorrect operation are easily found.

To speed operation, COPMON allows the operator to specify the information that will be printed out when a breakpoint, single step or trace occurs. This is done with the AUTOPRINT command, especially useful during single-step operation. The COPS registers and RAM locations can also be examined and modified directly with MODIFY. The program in shared memory can be changed with ALTER or PUT.

Another major function available on COPMON is the TIME command. This can be used to determine the time, in milliseconds, between two specified trigger conditions. (A trigger condition can be an address or any combination of the external event lines EXT1 and EXT2.)

## 4.2 Console Operation

To call COPMON from the STARPLEX console, the STARPLEX COPS ISE Software diskette should be loaded into disk drive #1 with a standard STARPLEX OS diskette in drive #0. The program to be loaded, COPMON, is then entered on the keyboard followed by a carriage return (CR). The system will respond:

```
>FDS1:COPMON
COPMON, REV:A, (DATE)
CHIP NUMBER (DEFAULT=420)?XXX
```

### NOTE

*In discussing any man/computer dialogue, there is a tendency to confuse "who said what." To avoid this problem, this manual will adhere to the policy of underscoring the operator-entered characters and the computer response will be printed without an underscore.*

The operator must enter a chip number from Table 4-1 in response to the system query. The chip number is used by COPMON to select the correct instruction subset, memory size, and register size. If no number is entered after the chip number prompt, COPMON defaults to the COP420 number. The chip number may also be changed later with the CHIP command. After the operator responds to the initial chip number prompt, COPMON responds with the COPMON prompt symbol, "<C>".

Example:

```
CHIP NUMBER (DEFAULT = 420)? 444
CHIP BEING EMULATED: COP 444
<C>
```

COPMON responds with the prompt after completing the execution of each command.

The following general rules apply to the console commands:

1. Numbers. COPMON syntax uses both decimal and hexadecimal numbers (see Table 4-3). Input from the user is treated as decimal or hexadecimal depending on what COPMON is expecting. If COPMON expects a decimal number it assumes that the user will enter a decimal number. Hexadecimal numbers do not require a leading zero; however, they do no harm since they are ignored. F3 is a valid hex number. The usual conventions for hex, an "H" at the end of a hexadecimal number (3FH) or an "X" at the beginning of a hex number (X'1F) are illegal.
2. Console Output. Console output of COPMON is normally sent to the CRT. The output of any one command may be directed to the printer by appending "LPT:" to the end of the command. (The "LPT:" must be immediately followed by a carriage return.) Example: C> STATUS LPT:  
The status now appears on the printer, instead of the CRT.  
Console output (whether to the CRT or line printer), may be interrupted at any time by pressing any key. Asterisks (\*\*\*\*\*) will be printed to indicate this.
3. Disk Files. The LOAD, COMPARE and SAVE commands use disk files. The default extension assumed is ".REL". If no device is specified, the STARPLEX default device "FDSx:" is assumed.

For convenience, both the COPMON and MASKTR programs can be copied onto the STARPLEX OS diskette. Drive #1 can then be used solely for user object programs.

#### 4.2.1 Dual Processor Emulation

Users of the dual processor COPS™ chips (COP2440, COP2441, and COP2442) should note the following points before attempting emulation.

1. The two processors are referred to as the X and Y processors. Processor X starts execution at address 0H on power-on, processor Y at address 401H.
2. COPMON makes sure that the two processors are always synchronized, that is, they execute instructions in the same order as they would if there were no breakpoints. While single-stepping, it is sometimes necessary for one processor to execute two or more instructions before the other executes any (for example, if one processor is breakpointed on a skipped 2-byte instruction).

It is possible for this synchronization to be lost, though it should not happen under normal circumstances.

When the Program Counters are printed, an asterisk (\*) is used to mark the PC of the processor which will execute next.

3. The hardware places some restrictions on triggering from a reset state. The ISE (target) board synchronizes when processor Y sends out address 401H. If the trigger condition becomes valid before this, correct synchronization is uncertain. For example, if a TRACE IMMEDIATE is performed from RESET, processors X and Y may get interchanged: i.e., processor X will be displayed on the right hand side of the screen, instead of the left hand side as usual. There is a 50-50 chance of this happening.

This uncertainty also exists if an External Event condition is used for TRACE, BREAKPOINT or TIME operations starting from RESET and the condition is valid before address 401H appears.

AS FAR AS POSSIBLE, SUCH UNCERTAIN TRIGGERING CONDITIONS SHOULD BE AVOIDED. IF SUCH AN OPERATION HAS TO BE PERFORMED, THE COP CHIP SHOULD BE RESET AFTER THE OPERATION, SINCE FURTHER EMULATION MAY BE INCORRECT.

4. COPMON operates in three basic modes, referred to as the DUAL, X-only and Y-only modes. The DUAL mode is the default, 'normal' mode of operation. The X-only and Y-only modes make it simpler to concentrate on the behavior of one particular processor and temporarily ignore the other. Refer to the 'SET PROCMODE' (Section 4.3.22) command for details.

#### 4.2.2 Documentation Conventions

The following documentation conventions are used in describing the command syntax. Upper-case and lower-case letters are used in these conventions; any combination of upper-case and lower-case letters may be used when actually entering the commands.

UPPER-CASE letters show the commands and parameter names such as key words, logical device names, switches, and options. Mandatory items are shown outside of the brackets <>, { }, and [ ]; they must be included in the command strings.

If an item shown consists of underscored letters followed by non-underscored letters, then that item may be entered in an abbreviated form. Minimum legal abbreviation of such items is the underscored letters portion; in addition, any number of the non-underscored letters that follow may also be used.

Spaces or blanks, when present in command strings, are significant; they must be entered as shown. However, multiple blanks may be used in place of a single blank and only one blank may be used in place of multiple blanks.

< >-angle brackets enclose descriptive names (in lower-case) for user-supplied names/labels for commands, parameters, devices, and files.

{ }-braces enclose more than one item out of which one, and only one, must be used. The items are separated from each other by a logical OR sign "||".

[ ]-brackets enclose optional item(s). Brackets within a bracket enclose item(s) which may be optionally entered only if the item outside that inner bracket is entered.

|-logical OR sign separates items out of which one and only one may be used.

. . .-three consecutive periods indicate optional repetition of the preceding item. If a comma precedes the three periods, then each item must be separated from the other by a comma.

### 4.3 COPMON Console Commands

The COPMON console commands are summarized in Table 4-2 and are described in detail here. Commands may be abbreviated to one or two characters as indicated by the underscored characters in Table 4-2 and in the syntax descriptions in the following discussion. Command options are defined in Table 4-3.

#### 4.3.1 ALTER SHARED MEMORY Command

Syntax: ALTER [<addr>] [<value>] . . .

This command alters the contents of consecutive shared memory locations to the specified hexadecimal values beginning at the specified address. Consecutive commas will increment the current address pointer, leaving the data at these locations unaltered. If no address is specified, the command begins at the last altered or listed location (see LIST command). If two or more values separated by spaces are given for <value>, the last of these values will be the one stored. The alterable range of shared memory is determined by the chip number. The COP chip is reset if it was running.

Example:

*C > A 1CF,D0,D1 ← Places in D0 location 1CF, leaves D0 unchanged, and places D1 in location 1D1.*

#### 4.3.2 AUTOPRINT Command

Syntax: AUTOPRINT [<print opt>] [<print opt>] . . .

The AUTOPRINT command is used to specify the information that will be printed when the COPS chip encoun-

ters a breakpoint, is single-stepped, or executes a trace. Table 4-3 lists all of the allowable print options. The default value is ALL which sets all of the applicable options on except S and ST. Some of the print options are only valid for breakpoints and single-steps; others are valid for trace operations. An "LPT:" entered at the end of the line will cause the autoprnt output to go to the printer instead of the console. The 16-digit contents of any specified RAM register will be printed, left to right, most-significant digit to least-significant digit.

Example:

```
C> AU A, P
```

causes the contents of the accumulator and the program counter to be printed after each breakpoint and single-step operation.

If it is desired to modify the current list of print options, a "+" or "-" may be placed in front of the list of options. In this case, ALL may not be used as a print option.

Example:

```
C> AU A,P ← Accumulator and program counter
      put in print option list
```

```
C> AU +M2 ← Now memory register 2 is also printed
      along with the accumulator and program counter.
```

If no <print opt>'s are specified, the autoprnt feature is turned off.

Example:

```
C> AU ← AUTOPRINT off
```

COP2440, 2441, 2442 users should refer to the 'SET PROCMODE' command (section 4.3.22) for changes in <print opt>'s with the default processor setting.

### 4.3.3 BREAKPOINT Command

```
Syntax: BREAKPOINT [<cond>|<cond> . . .]
      [,<occur#>[,<gopt>]]
```

The BREAKPOINT command sets the breakpoint enable flag and establishes the conditions that will cause breakpoints to occur. Up to ten conditions can be specified, but only the first will be monitored. When that condition is satisfied and a breakpoint executed, the list of conditions is rotated so the next condition on the list becomes the one being monitored. If the BREAKPOINT command is entered with no conditions specified, all previous conditions are retained. If the BREAKPOINT command is entered with one or more conditions, all of the previous conditions are cleared and replaced by the new ones contained in the command string. If the occurrence number is not specified, the system defaults to the last specified value. If <gopt> is specified, the breakpoint operation occurs repeatedly on successive conditions in the circular list. This continues until a break is received from the console. When the breakpoint occurs, the data specified earlier by the AUTOPRINT command is printed out to provide the operator with a snapshot of the pertinent data during the COPS program execution.

During a breakpoint, the system automatically does a trace with a prior count of 240. This information about the 240 cycles prior to the breakpoint may be printed using the TYPE command. Locations corresponding to the breakpointed state of the chip are displayed as asterisks ("\*\*\*\*\*").

The BREAKPOINT command sets the breakpoint enable flag but does not actually initiate the breakpoint. That is done by the next GO command which initiates program execution. Since the breakpoint operation occurs from the shared memory on the ISE Board, if the operator is running from programs contained in PROMs on the Emulator Board, the shared memory must contain the same data as the PROMs.

Example:

```
C> BR 2/35//EVX1/26, 4, G
```

```
BREAKPOINT ENABLED
```

```
A:2 A:35 IMED EVX1 A:26 OCCUR:4 GO:Y
```

Break flag is enabled, the next GO will cause successive breakpoints on the fourth occurrence of each of the five conditions, circling through the list until interrupted.

COP2440, 2441, 2442 users should refer to the 'SET PROCMODE' (section 4.3.22) command for changes in <cond> with the default processor setting. Also, during a breakpoint, both processors are traced, even if the mode is X-only or Y-only.

### 4.3.4 CLEAR Command

Syntax: CLEAR

The CLEAR command clears the breakpoint enable, trace enable, and time enable flags. The conditions associated with each of these functions remains unchanged.

Example:

```
C> C
```

```
BREAKPOINT, TRACE, AND TIME DISABLED
```

### 4.3.5 CHIP Command

Syntax: CHIP <chip#>

The CHIP command allows the operator to change and display the current chip number. Since the chip number determines the memory and register size, this must be done prior to emulating a COPS™ chip. See Table 4-1. If no chip is specified, the current chip number is displayed.

Example:

```
C> CH 444
```

```
CHIP BEING EMULATED: COP 444
```

Example:

```
C> CH
```

```
CHIP BEING EMULATED: COP 444
```

If the chip being emulated is a COP410 or a COP 411, COPMON will respond with another query:

```
ROMLESS PART (DEFAULT = 401)?
```



The operator must enter either of the following: 401, 402 or 404 depending on which ROMless part is being used on the Emulator Board (COP401L, COP402 or COP404L).

Example:

```
CHIP NUMBER (DEFAULT = 420)? 411
ROMLESS PART (DEFAULT = 401)? 402
CHIP BEING EMULATED: COP 411
ROMLESS PART BEING USED: COP 402
C>
```

#### 4.3.6 COMPARE Command

Syntax: COMPARE <filename>

The COMPARE command checks the load module on disk against the shared memory on the ISE™ Board. Each pair of values that does not compare is displayed. This continues until either the entire file has been examined or a break is received from the console. The COP chip is reset.

Example:

```
C> CO FDS1:DEMO
003 S:00 F:3C 057 S:8A F:B3 ← S: indicates shared
memory, F: indicates a disk file, and 003 indicates
an address.
COMPARE DONE
```

Note: Only those shared memory locations which are defined in the load module are compared.

#### 4.3.7 DEPOSIT Command

Syntax: DEPOSIT <value>, <addr range>

This command puts the specified value into each location of the specified address range. If the COP chip is running, it will be reset.

Example:

```
C> D F6, 11/1E ← F6 is put in locations 11 through
1E of shared memory.
```

#### 4.3.8 END Command

Syntax: END

Exits from COPMON and returns control to the STARPLEX™ Operating System. Pressing the END Key on the keyboard also has the same effect.

Example:

```
C> END
```

#### 4.3.9 FIND Command

Syntax: FIND <value>[,<addr range>[,<mask>]]

The FIND command searches the shared memory for the specified value and each occurrence is printed out. If the mask option is present, each shared memory byte is ANDed with the value of <mask> before it is tested. This allows the user to search out specific portions of bytes. If the mask option is not specified, it defaults to OFFH. Each occurrence of value is printed

on the console until the search is done or it is interrupted from the console. If the COP chip is running it will be reset.

Example:

```
C> F 8E,200/3FF
2CC:8E 2B0:8E 3FF:8E
FIND DONE
```

If the <value> typed in is three characters or more, a 2-byte search is performed. This is useful for locating 2-byte instructions. In this case, the mask defaults to OFFFFF.

Example:

```
C> F 6310, 100/3FF
275:6310 372:6310
FIND DONE
```

#### 4.3.10 GO Command

Syntax: GO [<addr>]

GO [<addrx>][,<addy>] ← *Dual processor only, see note below.*

The GO command causes the COPS chip to go to a specified address and begin executing the program there. The details of exactly how this is done vary somewhat depending on the status of the chip and the breakpoint and trace enable flags. Generally speaking, a breakpoint will be initiated if the breakpoint enable flag is set, a trace will be done if the trace enable flag is set, a time operation will be done if the time enable flag is set, and the chip will be started in a normal manner if neither flag is set. See Table 4-4. Breakpoint and trace flags remain unchanged after the GO command. For example, if the breakpoint flag is enabled, the first condition on the list is EVOX, the autoprnt options are B, P, and <gopt> is not set, the following sequence will occur:

Example:

```
C> GO
BREAKPOINTED ON EVOX AT A:xxx
B:01 P:xxx
```

xxx indicates the address at which EVOX occurred. A similar message would appear if trace were enabled instead of breakpoint.

Note: For COP2440, 2441, 2442 users:

Two addresses can be specified when emulating dual-processor COPS.

<addrx> = address for processor X  
<addy> = address for processor Y

If the processor mode is X-only or Y-only (see 'SET PROCMODE' command), and a single address is specified, it is assumed to refer to the default processor.

Example:

```
C> SET PR Y ← set processor Y as default
C> G 58 ← will start processor Y at address 58
C> G 27,439 ← start processor X at 27, processor
Y at 439
```

#### 4.3.11 HELP Command

Syntax: HELP

The HELP command causes a summary of the COP-MON commands to be printed on the console. The HELP key on the STARPLEX keyboard has the same effect.

#### 4.3.12 LIST Command

Syntax: LIST [<addr range>[,<addr range> . . .]]

The LIST command lists the contents of the shared memory across the specified address ranges. Each range printed begins at the next lower multiple of X'10. If <addr range> is just one value, only the contents of that location are printed. If no address range is specified, 256 locations are listed starting from the multiple of X'10 below the current address. The current address is the last address printed or altered. Subsequent LIST commands with no operands will list the next 256 locations. The COPS™ chip is reset only if it was running when the LIST command was issued.

Example:

```
C> L 4/8
000 00 C2 00 F2 03 29 76 AA D0
```

#### 4.3.13 LOAD Command

Syntax: LOAD <filename> [0]

This command loads the specified load module into shared memory. If the optional "0" (for 'Overlay') is present in the command string, the shared memory will not be cleared out first. LOAD automatically resets the COPS chip.

Example:

```
C> LO DEMO
FINISHED LOADING
```

#### 4.3.14 MODIFY Command

Syntax: MODIFY <print opt>, <value>[,<value1> . . .]

The MODIFY command is used to change the registers on the COPS chip. Since these registers include the I/O ports as well as the general purpose registers and RAM registers, the MODIFY command can be used to debug a hardware prototype system prior to the prototype software being completed. Each MODIFY command is used to change a single register on the chip. The MODIFY command is valid only while breakpointed.

Example:

```
C> BR 1
BRKPT ENABLED
A:001 OCCUR 1 GO:
C> R
CHIP IS RESET
C> GO
BREAKPOINTED ON A:001 AT A001
```

```
C> M M0, 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F ← This com-
mand sets memory register 0 digit 0 to 0, memory
register 0 digit 1 to 1, etc.
```

```
C> M M15,5,6,7,8 . . . ← This command sets memory
register 1 digit 5 to 5, etc.
```

```
C> M E,4 ← This command loads the E register with
4 (enable Q register to L bus).
```

```
C> M Q,AA ← This command, in conjunction with the
previous command, loads the Q register with AA and
thus puts AA on the L bus.
```

```
C> M D,B ← This command puts a HEX B on the D
port. Bits 0, 1, 3 are high and bit 2 is low.
```

```
C> M B,3D ← This command sets the B register to
RAM address 3,13.
```

COP 2440, 2441, and 2442 users should refer to the 'SET PROCMODE' command (Section 4.3.22) for changes in <print opt>'s with the default processor setting.

#### 4.3.15 NEXT Command

Syntax: NEXT [<gopt>]
NEXT [<gopt>] | X[,<gopt>] | Y[,<gopt>] ← Dual
processor only. See 'SET PROCMODE'
command.

This command is identical to the SINGLESTEP command (see Section 4.3.18), except at a JSR or JSRP instruction, where it will set a breakpoint at the instruction immediately after the JSR/JSRP and breakpoint thereafter executing the subroutine in real-time.

#### 4.3.16 PUT Command

Syntax: PUT [<addr>][,<instruct>[,<instruct>]]

The PUT command replaces the contents of the shared memory beginning at the address specified with the opcodes of the specified instruction mnemonics. If no address is given, placement begins at the current address. This command resets the COPS chip if it is running. Instruction opcodes may be directly specified in the operand field. Instructions with double operands may only be specified in hex format and, unlike the assembler format, double operands may not be separated by commas (e.g., LBI 23 is OK; LBI 2, 3 is not allowed).

Example:

```
C> P 130, CLRA, AISC 5, LBI 39
C>
```

#### 4.3.17 RESET Command

Syntax: RESET

This command resets the COPS chip and sets the reset flag, which in turn determines the operation of the GO command. See Table 4-4.

Example:

```
C> R
CHIP IS RESET
```

#### 4.3.18 SINGLESTEP Command

Syntax: SINGLESTEP [<gopt>]
SINGLESTEP [<gopt>] | X[,<gopt>] | Y
[,<gopt>] ← Dual processor only. See 'SET
PROCMODE' command.

The SINGLESTEP command performs a breakpoint on the next instruction. If the COPS chip is reset, it breakpoints at address 1. If it has already breakpointed, it steps one instruction. After each single step, information specified in the AUTOPRINT command is printed. If <gopt> is included, it will automatically step and print data until interrupted by the console.

If the COP chip is breakpointed, a carriage return is identical to single step without <gopt>.

Example:

```
C> S G ← Go immediately after printing.
```

```
(Step)
```

```
A:0 P:10 51 AISC 1
```

```
(Step)
```

```
A:1 P:11 53 AISC 3
```

```
.
```

```
.
```

```
.
```

#### 4.3.19 SAVE Command

Syntax: SAVE <filename>

This command saves the contents of shared memory in the specified file. All of shared memory, from address 0 to the maximum address of the chip being emulated, is saved. Shared memory itself is unchanged. This file may later be loaded back into shared memory using the LOAD command. The COPS™ chip will be automatically reset. The saved program *cannot* be used in MASKTR to generate a transmittal file.

Example:

```
C> SA MYPROG.002
```

```
SAVED MYPROG.002
```

#### 4.3.20 SEARCH Command

Syntax: SEARCH <addr>

This command searches the Trace memory for the specified address. Each occurrence is displayed and it searches until finished or interrupted by the console. Each line of output from the SEARCH command and the TYPE (trace memory) command contains the following information, from left to right:

1. Trace Memory location.
2. Location relative to TRACE condition location.
3. Program counter.
4. Skip Indication.
5. Value of external event inputs E4-E1, left to right.

Example:

```
C> SE 2FE
```

```
0 0 A:2FE SKIP E:1111
```

```
8 8 A:2FE E:1101
```

```
SEARCH DONE
```

#### 4.3.21 SET Command

Syntax: SET SIOMODE <Y/N>;  
SET STACKMODE <Y/N>

This command allows the user to turn the SIOMODE and STACKMODE flags on and off. The SIO register will be dumped during breakpoint and can be modified only if SIOMODE is on. Similarly, if STACKMODE is on, the stack will be dumped and displayed during breakpoint and single-step. The stack may also be modified.

There is one limitation in using STACKMODE. If the COP is breakpointed in an interrupt routine and STACKMODE is ON, the interrupt skip status flag in the COP chip may be lost. It cannot be restored. (If lost, a message will be printed.) This limitation does not apply to the COP440, 441, 442, 2440, 2441, 2442 and hence, for these chips, the default is STACKMODE ON?

Use of the SET command will automatically reset the COP chip and set AUTOPRINT to ALL.

Example:

```
C> SET ST Y
```

```
STACKMODE : Y
```

#### 4.3.22 SET PROCMODE Command (COPS 2440, 2441 and 2442 only.)

Syntax: SET PROCMODE {X|Y|D}

This command is used to set the default processor mode for dual processor emulation. The effects of setting a particular mode are best seen by example.

1. BREAKPOINT, TRACE and TIME <cond>s.

A hex address by itself refers to the default processor.

Example:

```
C> SET PR D ← Set 'DUAL' mode
```

```
C> BR 23 ← Breakpoint on address 23 of either processor
```

```
C> BR 23-X ← Breakpoint on address 23 of processor X
```

```
C> SET PR X ← Set 'X-only' mode (i.e., default is X)
```

```
C> BR 234 ← Breakpoint on address 234 of processor X
```

```
C> TR 23-Y ← Trace on address 23 of processor Y
```

```
C> TR 23-D ← Trace on address 23 of either processor
```

The default processor setting has no effect on External Event or Immediate triggering.

Example:

```
C> TR EVX1
```

This will initiate a trace when External Event 1 = 1, regardless of the default processor setting and regardless of the processor cycle during which the event is detected.

## 2. AUTO PRINT, TYPE and MODIFY &lt;print opt&gt;s

Example:

```
C> SET PR D ← Set 'DUAL' mode
C> AU AX,CY ← Will print AX and CY
C> MO A,3 ← Is ambiguous (Modify AX or AY?)
C> SET PR Y ← Set 'Y-only' mode (i.e., default is Y)
C> AU ALL ← Will print all RAM I/O registers, and
all processor Y registers (i.e., AY,CY, etc.)
C> AU A,BX,C ← Will print AY,BX,CY
C> MO A,3 ← Will modify AY to 3
```

## 3. SINGLESTEP and NEXT operations

Syntax: SINGLESTEP [<gopt>]|X[,<gopt>]|Y  
[,<gopt>]  
(or NEXT)

Example:

```
C> SET PR D ← Set 'DUAL' mode
C> S ← Singlestep on processor which is to
execute next
C> S G ← Singlestep continuously on alternate
processors
C> SET PR X ← Set 'X-only' mode (i.e., default
is X)
C> N ← Do a 'NEXT' on processor X
C> S Y ← Singlestep on processor Y
C> S G ← Singlestep continuously on
processor X
```

## 4. GO operation

Refer to 'GO' command description, Section 4.3.10.

As with other SET commands, the SET PROCMODE command will reset the COPS chip and restore default AUTOPRINT conditions. In addition, it will set BREAKPOINT, TRACE, and TIME conditions to their default values.

## 4.3.23 SHARED MEMORY Command

Syntax: SHARED MEMORY <Y/N>

This command allows the operator to specify whether the COPS chip runs from shared memory or the PROMs on the Emulator Board. If "Y" is entered, the chip will run from shared memory. If "N" is entered, the chip will run from the PROMs. The chip is automatically reset by this command.

Example:

```
C> SH Y
SHARED MEMORY MODE
C> SH N
PROM MODE
C>
```

## 4.3.24 STATUS Command

Syntax: STATUS

This command causes the status of the COPS chip and various internal conditions to be printed out.

Example:

```
C> ST
CHIP BEING EMULATED: COP420
CHIP IS RESET
BREAKPOINT, TRACE AND TIME DISABLED
SHARED MEMORY MODE
NO UNASSEMBLY
SIO REG MODE: N
STACK MODE: N
BRKPT CONDITIONS:
A:005 OCCUR: 1 GO:N
TRACE CONDITIONS:
EVX1 OCCUR:1 PRIOR:0 GO:N
TIME CONDITIONS:
A:001 OCCUR:1 A:237 OCCUR:2 GO:Y
```

## 4.3.25 TIME Command

Syntax: TIME [<cond1>[,<occur1>]  
[/<cond2>[,<occur2>[,<gopt>]]]]

The TIME command sets and prints the conditions which control the time measurement. This timer is started when the first set of conditions is met and the timer is stopped when the second set of conditions is met. The second set of conditions is invoked only after the first set of conditions is satisfied, and it is looked for from that time. If those conditions have been encountered prior to the first set of conditions having been met, they are ignored. If only cond1 is specified, cond2 is set to cond1 and occur2 is set the same as occur1. If cond1 and cond2 are specified, occur1 and occur2 are left at their previous values.

The time is reported in milliseconds. The limits on the TIME command are that the time between the events must be greater than 500µs and less than nine minutes. If the time is less than 500µs, the events may not be recognized, or if they are recognized, the time reported could be wrong. If the time is greater than nine minutes, a timer overflow message will be printed. The resolution of the TIME command is ±100µs.

As in the TRACE command, the TIME command is not initiated until a GO command is issued. The TIME, TRACE, and BREAKPOINT commands are mutually exclusive.

COP2440, 2441, and 2442 users should refer to the 'SET PROCMODE' command (Section 4.3.22) for changes in <cond> with the default processor setting.

**NOTE**

The TIME command operates by disabling interrupts on the STARPLEX™ CPU and maintaining a software timer based on CPU instruction execution times. THIS WILL TEMPORARILY HALT THE STARPLEX SYSTEM REAL-TIME CLOCK, but will not affect operation of the system in any other manner.

Example:

C> TI EVX1,2/234,3 ← This command will measure the time from the second positive transition on EXTERNAL EVENT 1 (high on 1, don't care on 2) to the third occurrence of address 234 after the EXTERNAL EVENT condition has been met.

TIME ENABLED

EVX1 OCCUR: 2 TO A:234 OCCUR: 3 GO:N

C> TI 350, 1/24,2,G ← This command will measure the elapsed time from the first occurrence of address 350 to the second occurrence of address 24 after the occurrence of address 350. It will repeat this until interrupted from the keyboard.

TIME ENABLED

A:350 OCCUR:1 to A:024 OCCUR:2 GO:Y

C> TI 44

TIME ENABLED

A:044 OCCUR:1 TO A:044 OCCUR:1 GO:N

C> GO ← This example shows the default conditions of the command. Used with the previous example, this command will measure the elapsed time between the first occurrence of address 44 and the next occurrence of address 44.

TIME FROM A:044 TO A:044 = 16.8 MS

#### 4.3.26 TYPE Command

Syntax: TYPE [print opt][,print opt ... ]]

The TYPE command prints out the information specified to the printer or console. As with the AUTOPRINT command, if a RAM register is specified, its 16-digit contents will be listed, from left to right, most significant digit to least significant digit. If no options are specified and a trace operation was just executed, trace memory will be displayed in blocks of 16. When printing trace memory while the chip is breakpointed, the last eight locations of trace memory will not be displayed.

Example:

C> T P, Q, B, M1F,M2

B:10 Q:FF P:004 OF LBI 0 M1F:0

M2:00000000120F120E

COP2440, 2441, and 2442 users should refer to the 'SET PROCMODE' command (Section 4.3.22) for changes in <print opt> with the default processor setting.

#### 4.3.27 TRACE Command

Syntax: TRACE [cond][,occur#][,prior][,gopt]]]

This command allows the user to set the print trace conditions. During a Trace operation, COPMON stores each consecutive value of the COPS™ program counter in a 254-word circular buffer, so that at any time during trace operating, the buffer has the previous 254 values of the program counter. The <cond> has been met the number of times specified by <occur#>, COPMON saves the number of values of the program counter to <cond> specified by <prior>, and fills the rest of the buffer with the subsequent values of the program counter. It then prints the <cond> specified and the address where <cond> was recognized, followed by any trace data specified by the AUTOPRINT command.

If <cond>, <occur#> or <prior> are omitted, they retain their previous values. If <gopt> is included, then each time a trace operation is finished, another GO command is performed with the same conditions continuing until interrupted by the console. The TRACE command does not initiate trace operation, but sets the Trace Enable flag so that trace operation is initiated on the next GO command. See Table 4-4.

Example:

C> TR EVOX, 2, 22

TRACE ENABLED:

EVOX OCCUR:2 PRIOR:22 G:N

Under certain conditions (see Table 4-4), the <prior> count specified may not be fulfilled. That is, <cond> may occur before <prior> cycles of the chip. In this case, when typing trace memory, a message of the form "ONLY nn prior LOCATIONS TRACED" will appear.

Example: Assume that all of shared memory contains NOP instructions, except location 0, which has a CLRA instruction.

C> R

CHIP IS RESET

C> AU A,P

C> S

STEP

A:0 P:001

C> TR 5,1,245

TRACE ENABLED

A:005 OCCUR:1 PRIOR:245 GO:N

C> G

TRACED ON A:005 AT A:005

C> T 0/250

#### Only Four Prior Locations Traced

241	-4	A:001	E:1111
242	-3	A:002	E:1111
243	-2	A:003	E:1111
244	-1	A:004	E:1111
245	0	A:005	E:1111
246	1	A:006	E:1111
247	2	A:007	E:1111
248	3	A:008	E:1111
249	4	A:009	E:1111
250	5	A:00A	E:1111

External Event lines, EXT4 to EXT1.

Program Counter value.

Location relative to <cond> (trace condition).

Location in Trace Buffer.

COP 2440, 2441, and 2442 users see 'SET PROCMODE' command for changes in <cond> with the default processor settings.

Also, when in DUAL mode, both processors are traced, and trace memory is restricted to locations 0 through 252. Processor X is displayed on the left hand side of the screen, processor Y on the right hand side.

If the mode is X-only or Y-only, only that processor is traced.

#### 4.3.28 UNASSEMBLE Command

Syntax: UNASSEMBLE { Y | N }

The UNASSEMBLE command mode will give an opcode and mnemonic for each instruction. This command selects the unassemble mode for use during trace and list operations. If a LIST is started on the second byte

of a two-byte instruction, the unassembly will be incorrect until two successive one-byte instructions are encountered.

Table 4-1. Valid Chip Numbers

CHIP #	Memory Size	RAM Register Address	RAM Digit Address
410/411	0-1FFH	0H-3H	0,9H-0FH
420/421/422	0-3FFH	0H-3H	0H-0FH
444/445	0-7FFH	0H-7H	0H-0FH
440/441/442	0-7FFH	0H-9H	0H-0FH
2440/2441/ 2442	0-7FFH	0H-9H	0H-0FH

**Note:** One of these numbers must be entered into the computer in response to the query for CHIP NUMBER? If no number is entered, COPMON will use the default chip number 420.

Table 4-2. Summary of COPMON Console Commands

Command Name	Operand Syntax	Description
<u>ALTER</u>	[<addr>][, [<value>] ...]	Alter Shared Memory
<u>AUTOPRINT</u>	[<print opt>][, <print opt> ...]	Set Print Options
<u>BREAKPOINT</u>	[<cond>[<cond> ...]][, <occur#>][, <gopt>]]	Set Breakpoint
<u>CLEAR</u>		Clear Trace and Breakpoint Flags
<u>CHIP</u>	<chip#>	Set or Display Chip Number
<u>COMPARE</u>	<filename>	Compare File with Shared Memory
<u>DEPOSIT</u>	<value>, <addr range>	Deposit Values into Shared Memory
<u>FIND</u>	<value>[, <addr range>][, <mask>]]	Find Value in Shared Memory
<u>END</u>		Exit COPMON
<u>GO</u>	[<addr>]	Begin Program Execution
<u>GO</u>	[<addrx>][, <addy>]	(Dual Processor Chips Only)
<u>HELP</u>		Display Command Summary
<u>LIST</u>	[<addr range>][, <addr range> ...]	List Shared Memory
<u>LOAD</u>	<filename> [O]	Load Shared Memory from File
<u>MODIFY</u>	<print opt>, <value>[, <value1> ...]	Modify Registers and COPS RAM
<u>NEXT</u>	[<gopt>]	Breakpoint on Next Instruction
<u>NEXT</u>	[<gopt>][X[, <gopt>]]Y[, <gopt>]	(Dual Processor Chips Only)
<u>PUT</u>	[<addr>][, <instruct>[, <instruct> ...]]	Put Instruction (Assemble)
<u>RESET</u>		Reset Chip
<u>SINGLESTEP</u>	[<gopt>]	Single-Step
<u>SINGLESTEP</u>	[<gopt>][X[, <gopt>]]Y[, <gopt>]	(Dual Processor Chips Only)
<u>SAVE</u>	<filename>	Save Shared Memory into File
<u>SEARCH</u>	<addr>	Search for Address in Trace Memory
<u>SET</u>	SIO { Y/N } or ST { Y/N }	Set SIOMODE or STACKMODE Flags
<u>SET</u>	PR <proc>	Set Default Processor Mode. Dual Processor Only
<u>SHARED MEM</u>	{ Y/N }	Set/Clear Shared Memory Mode
<u>STATUS</u>		Display Chip Status
<u>TIME</u>	[<cond1>[, <occur1>]][<cond2>[, <occur2>][, <gopt>]]]	Measures Elapsed Time
<u>TYPE</u>	[<print opt>][, <print opt> ...]	Type Breakpoint or Trace Data
<u>TRACE</u>	[<cond>[, <occur#>[, <prior>[, <gopt>]]]]	Set Trace Conditions
<u>UNASSEMBLE</u>	{ Y/N }	Display Instruction Mnemonics of the Data in Shared Memory

Table 4-3. Operand Syntax

Operand	Description
<addr>	One to three hexadecimal digits, < = maximum address of the chip defined by <chip#>. P = Previous address. . = Current address, i.e., last address altered or typed. N = Next address. L = Last address defined by chip number.
<addr cond>	Address in hexadecimal, greater than 0, less than or equal to maximum address of chip.
<addr range>	<addr>[/<addr>]
<chip#>	410, 411, 420, 421, 422, 444, 445, 440, 441, 442, 2440, 2441 or 2442
<cond>	<addr cond> <evt cond> I = immediate trace or breakpoint. Cannot use with TIME command. <addr cond>-<proc> (Dual processor only).
<dig#>	Hexadecimal digit specifying RAM digit address, see Table 4-1.
<end>	Decimal 0-253; last location of trace memory desired. (See Note 1.)
<evt cond>	EV00, EV01, EV10, EV11, EVX0, EVX1, EV0X or EV1X. Format: EV<EXT2><EXT1> '1' = Logic 1 '0' = Logic 0 'X' = Don't care
<filename>	Valid STARPLEX filename, default extension assumed is .REL.
<gopt>	G = Go immediately after printing.
<instruct>	Valid COPS instruction mnemonic with hexadecimal with operand. The operand is hexadecimal.
<mask>	Hexadecimal 0-0FFH. (0-0FFH for a 2-byte FIND).
<occur#>	Decimal 1-256. Number of times <cond> occurs before initiating BREAKPOINT, TRACE or TIME.
<print opt> (See Note 2)	A = Accumulator (BR,M) ALL = All breakpoint data (BR) B = RAM address register B (BR,M) C = Carry bit (BR,M) D = Output port (M) E = EN register (M) (See Note 3.) G = G I/O port (BR,M) H = H I/O register (BR,M) I = I Input port (BR) L = L I/O port (BR)
<print opt> (cont'd)	M = All RAM on chip (BR) M<reg#> = RAM Register <reg#> (BR,M) M<reg#><dig#> = RAM digit <reg#><dig#> (BR,M) N = N (stack pointer) register (BR,M) P = Program counter (BR) R = R I/O register (BR,M) S = Serial I/O register (only if SIOMODE is true) (BR,M) SA = Stack register SA. -----: SB = Stack register SB. ---: (BR,M) SC = Stack register SC. : See Note 4. SD = Stack register SD. -----: ST = All stack registers (only if STACKMODE is true) (BR) T = Trace memory 0 through 253 (BR,TR) (See Note 2.) TI = T (Timer) register (BR,M) <start> = Trace memory location <start>. (BR,TR) <start>/<end> = Trace memory <start> through <end>. (BR,TR) AX,BX,CX,NX = A,B,C & N registers of processor X (BR,M) AY,BY,CY,NY = A,B,C & N registers of processor Y (BR,M) PX = Program Counter, Processor X (BR) PY = Program Counter, Processor Y (BR) SAX,SBX,SCX,SDX = Stack registers of Processor X (BR,M) SAY,SBY,SCY,SDY = Stack registers of Processor Y (BR,M) STX = All stack registers of Processor X (BR) STY = All stack registers of Processor Y (BR)

Table 4-3. Operand Syntax (continued)

Operand	Description
<prior>	Decimal 0-253, number of addresses traced prior to <cond> (See Note 2.)
<proc>	X Y D designates processor X, Y, or Dual.
<reg#>	Hexadecimal digit specifying RAM register.
<start>	Decimal 0-253; first location in trace memory desired (See Note 2.)
<value>	Hexadecimal 0-0FFH

**Note 1:** If using a Dual processor COPS in Dual mode, the maximum value is limited to 252.

**Note 2:** Print options marked with (BR) apply to breakpoint and singlestep operations, those marked (TR) apply to trace operations, and those marked (M) apply to the Modify command.

**Note 3:** Also applies to breakpoint and singlestep (BR) for COPS 440, 441, 442, 2440, 2441 and 2442.

**Note 4:** Valid only if STACKMODE is true. Also, for COPS 440, 441, 442, 2440, 2441 and 2442, if not a valid stack entry as indicated by the stack pointer reg N, a '?' is printed after the entry. For example, on the COP 440, if N = 1, SA and SB are printed as SA:023 SB:344?

Table 4-4. GO Operation Summary

Address Given	BRKPT or TRACE Enabled	COP Chip Status	Function Performed
No	No	Reset	Start chip at addr 000.
No	No	Breakpointed	Start chip at BRK addr.
No	No	Running	"COP ALREADY RUNNING."
No	BRKPT	Reset	Start chip at addr 000, enter breakpoint mode.
No	BRKPT	Breakpointed	Start chip at BRK addr, enter breakpoint mode.
No	BRKPT	Running	Enter the breakpoint mode.
No	TRACE	Reset	Start chip at addr 000, enter trace mode.
No	TRACE	Breakpointed	This is allowed, but the prior count condition specified by the user in enabling TRACE may not be fulfilled.
No	TRACE	Running	Enter TRACE mode.
Yes	No	Reset	Breakpoint at 1, start chip at (ADDR).
Yes	No	Breakpointed	Start chip at (ADDR).
Yes	No	Running	"COP ALREADY RUNNING."
Yes	BRKPT	Reset	Breakpoint at 1, start chip at <addr>, enter breakpoint mode.
Yes	BRKPT	Breakpointed	Start chip at <addr>, enter breakpoint mode.
Yes	BRKPT	Running	Breakpoint immediate, start chip at <addr>, enter breakpoint mode.
Yes	TRACE	Reset	Breakpoint at 1, start chip at <addr>, enter trace mode.
Yes	TRACE	Breakpointed	This is allowed, but the prior count condition specified by the user in enabling TRACE may not be fulfilled.
Yes	TRACE	Running	Breakpoint immediately, start chip at <addr>, enter trace mode.

**Note:** The function of the GO command depends on the mode that the COPS™ chip is in whether or not BRKPT or TRACE or Time is enabled, and whether or not <addr> is given.

The TIME enable flag has the same effects as the TRACE flag, i.e., if TIME is enabled, just substitute TIME for TRACE in the table.



Table 4-5. Keyword Abbreviations

Keyword	Minimum Legal Abbreviation	Keyword	Minimum Legal Abbreviation
ALTER	A	NEXT	N
AUTOPRINT	AU	PUT	P
BREAKPOINT	B	RESET	R
CLEAR	C	SINGLESTEP	S
CHIP	CH	SINGLESTEP	S
COMPARE	CO	SAVE	SA
DEPOSIT	D	SEARCH	SE
FIND	F	SET	SET
END	END	SET	SET
GO	G	SHARED MEM	SH
GO	G	STATUS	ST
HELP	H	TIME	TI
LIST	L	TYPE	T
LOAD	LO	TRACE	TR
MODIFY	M	UNASSEMBLE	U
NEXT	N		

# Mask Transmittal Program (MASKTR)

## 5.1 MASKTR

After all of the program writing, software bug hunting, hardware glitch locating, and so on, we are at last ready to turn our software into ROM masked COPS™ chips. To do this, we need a way of transmitting our program to National Semiconductor so that the appropriate masks can be fabricated and the custom COPS chips built. Of course, we could translate our program into pencil marks on cards. In the early days of ordering ROMs, this was exactly what was done. But the cost of an error is too high, and with 4,000 bytes of program to contend with, there is a very high likelihood of introducing an error.

The COPS ISE™ Subsystem solves the problem with another utility program, located on the COPS System Diskette. That program, MASKTR, accepts object code load modules prepared with the COPS Assembler as inputs and translates them into a standard format that can be stored on a second diskette which is sent to National. That program is the subject of this chapter.

MASKTR works with two files: an input file called the Load Module and a second file created from the Load Module called the Transmittal File. The Transmittal filename is the same as the Load Module filename, but the modifier is .TRN. The Transmittal File contains the following information:

1. Name and phone number of the responsible person.
2. Company name and address.
3. Date.
4. Chip Number.
5. Listing of options showing option number, option name and option value.
6. ROM data including addresses, unused addresses are set to opcode zero (0) which is a CIA instruction.
7. Source, object, and Transmittal file checksums.

To enter any information for the Transmittal file, MASKTR must first be in the Transmittal mode. This mode may be entered with the Transmittal command (T) followed by the load module filename. The default modifier is .REL.

When MASKTR is in the Transmittal mode, the user is requested to provide the following information:

1. Chip number.
2. Name and phone number of responsible person.
3. Company name and address.
4. Date.
5. Option values.

MASKTR prompts the user with a description of the desired item required by the program, the current value of the data item (as last entered by the user or specified in the load module), and then asks for the new value from the user. If no change is required, a carriage return will leave the value unchanged. If a change is requested for the options, the value entered is checked for validity. Entering a blank line causes an advance to the next item to be entered.

To execute MASKTR, type:

```
C> :MASKTR
MASKTR, Rev:B, (Date)
T>
```

MASKTR uses a T> as a prompt. When it appears on the console, any of the MASKTR commands summarized in Table 5-1 can be entered.

MASKTR can be entered directly from the Command Interpreter of the STARPLEX™ OS. It is entered in the same fashion as any other STARPLEX software utility.

## 5.2 MASKTR Console Commands

### 5.2.1 ABORT Command

Syntax: ABORT

This command aborts the creation of a Transmittal file and returns control to the Prompt mode.

Example:

```
T> A
ABORT TRANSMITTAL FILE CREATION
(Y/N, CR = YES)CR
TRANSMITTAL FILE ABORTED
T>
```

### 5.2.2 COMPANY Command

Syntax: COMPANY

The COMPANY command causes MASKTR to prompt the user for the company name and address. Eight lines are allowed for this entry.

Example:

```
T> CC
COMPANY NAME AND ADDRESS:
UNSPECIFIED
COMPANY NAME AND ADDRESS:
NATIONAL SEMICONDUCTOR
2900 SEMICONDUCTOR DRIVE
SANTA CLARA, CA 95051
CR
DATE: UNSPECIFIED
```

### 5.2.3 DATE Command

Syntax: DATE

The DATE command causes MASKTR to prompt the user for the date. One line is allowed for this entry.

Example:

```
T> D
DATE: UNSPECIFIED
DATE: 1 JANUARY, 1980
OPTION 1 GROUND = 0
```

### 5.2.4 ERROR Command

Syntax: ERROUR <LPT:>

This command summarizes any option conflict which must be resolved before the Transmittal file may be created. This summary may be directed to the printer by including "LPT:" at the end of the command line.

Example:

```
T> E
ILLEGAL CKO, CKI COMBINATION, CKO = 0,
CKI = 4
T>
```

### 5.2.5 FINISH Command

Syntax: FINISH

The FINISH command finishes the creation of the Transmittal file, and writes it onto the disk. If all of the options have been defined, the system will prompt the user to insert a diskette that will receive the newly created Transmittal File and will presumably be sent to National. This disk must be a formatted disk, i.e., a disk formatted with the standard FORMAT command in the STARPLEX™ Utilities.

Example:

```
T> F
(Y/N,CR = YES)? CR
DISK TO BE MAILED IN DRIVE FDS1:
(Y/N,CR = YES)? CR
CREATING FILE FDS1: xxxx.TRN
T>
```

### 5.2.6 LIST Command

Syntax: LIST

The LIST command lists the Transmittal file as it will appear on the form returned to you from National for verification and sign-off before the mask is generated. A "LPT:" at the end of this command line will cause the listing to go to the system printer.

Example:

```
T> L
```

This example will list the Transmittal file on the console. The listing may be interrupted by any key-stroke. The user may then either continue the listing or return to the Prompt mode.

### 5.2.7 NAME Command

Syntax: NAME

The NAME command prompts the user for the name/phone number of the person responsible for this program. Two lines are allowed for this entry.

Example:

```
T> N
RESPONSIBLE NAME/PHONE:
UNSPECIFIED
RESPONSIBLE NAME/PHONE:
JOE USER
123 456 7890
COMPANY NAME/ADDRESS:
UNSPECIFIED
```

### 5.2.8 Option Command

Syntax: OPTION <opt#>

This command causes the program to prompt the user for the valid options for the chip specified. If the opt# is omitted, the program prompts for options from the first option.

Example:

```
T> O 12
OPTION 12: L3 DRIVER = UNSPECIFIED
00 = STANDARD OUTPUT
01 = OPEN DRAIN
02 = HI CURRENT LED SEG OUT
03 = CURRENT TRI-STATE
04 = LOW CURRENT LED SEG OUT
05 = LOW CURRENT TRI-STATE
OPTION 12: L3 DRIVER 01
OPTION 13: L2 DRIVER = UNSPECIFIED
```

### 5.2.9 PRINT Command

Syntax: PRINT <chip#>

The PRINT command prints out the allowable options for the chip specified in the command. If a LPT: is entered at the end of the command line, the options are sent to the printer instead of the console. The PRINT command cannot be used while in the Transmittal mode.

Example:

```
T> P 420
CHIP NUMBER:420
OPTION 1: GROUND
NOT AN OPTION
OPTION 1: CKO OUTPUT
00 = CLOCK GEN OUT XTAL/RES
01 = RAM KEEP ALIVE
02 = GENERAL INPUT, VCC LOAD
03 = MULTICOP SYNC IN
04 = GENERAL INPUT, HI-Z
```

### 5.2.10 TRANSMITTAL Command

Syntax: TRANSMITTAL <filename>

When the TRANSMITTAL command is invoked, the chip number prompt is given. The Load Module is read into memory, and the entered chip number is checked against the chip number contained in the Load Module. If the chip numbers are not compatible, MASKTR aborts the Transmittal command and returns to Prompt mode. If the chip numbers are compatible, the valid chip number is entered into the data table and used to determine which options are valid and available. The ROM data and option values (if any) from the Load Module are also entered into the data table.

The filename specified in the TRANSMITTAL command must include the drive specification.

The examples for Transmittal are included in the next section which is a sample working session for MASKTR.

### 5.3 MASKTR Example

The easiest way to get a feeling for MASKTR is to follow a sample workout. In this section, a Load Module named MASKEK is to be transmitted to National. First, MASKTR itself must be called into the STARPLEX™ system.

```
C> :MASKTR
MASKTR, Rev:B, (Date)
T> T FDS1:MASKEK
CHIP NUMBER: 421
LOAD MODULE CHIP NUMBER ERROR
T> T FDS1:MASKEK
CHIP NUMBER: 420
RESPONSIBLE NAME/PHONE:
UNSPECIFIED
RESPONSIBLE NAME/PHONE:
JOE COPUSER
(415) 777-6234
COMPANY NAME/ADDRESS:
UNSPECIFIED
COMPANY NAME/ADDRESS:
NATIONAL SEMICONDUCTOR
2900 SEMICONDUCTOR DRIVE
SANTA CLARA, CA 95051
CR
DATE: UNSPECIFIED
DATE: JANUARY 5, 1979
OPTION 01: GROUND = 00
    NOT AN OPTION
OPTION 02: CKO OUTPUT = 02
    00 = CLOCK GEN OUT XTAL/RES
    01 = RAM KEEP ALIVE
    02 = GENERAL INPUT, VCC LOAD
    03 = MULTICOP SYNC IN
    04 = GENERAL INPUT, HI-Z
OPTION 02: CKO OUTPUT CR
OPTION 03: CKI INPUT = 04
    00 = XTAL/16
    01 = XTAL/8
    02 = TTL/16
    03 = TTL/8
    04 = RC/4
    05 = EXT OSC/4
OPTION 03: CKI INPUT CR
```

```
OPTION 04: RESET INPUT = 00
    00 = LOAD VCC
    01 = HI-Z
OPTION 04: RESET INPUT 1
OPTION 05: L7 DRIVER = 02
    00 = STANDARD OUTPUT
    01 = OPEN DRAIN
    02 = CURRENT LED SEG OUT
    03 = HI CURRENT TRI-STATE
OPTION 05: L7 DRIVER CR
OPTION 06: L6 DRIVER = 02
    00 = STANDARD OUTPUT
    01 = OPEN DRAIN
    02 = HI CURRENT LED SEG OUT
    03 = HI CURRENT TRI-STATE
OPTION 06: L6 DRIVER CR
OPTION 07: L5 DRIVER = 02
    00 = STANDARD OUTPUT
    01 = OPEN DRAIN
    02 = HI CURRENT LED SEG OUT
    03 = HI CURRENT TRI-STATE
OPTION 07: L5 DRIVER CR
OPTION 08: L4 DRIVER = 02
    00 = STANDARD OUTPUT
    01 = OPEN DRAIN
    02 = HI CURRENT LED SEG OUT
    03 = HI CURRENT TRI-STATE
OPTION 08: L4 DRIVER CR
OPTION 09: IN 1 INPUT = 02
    UU = 1 IL LOAD
    01 = TTL HI-Z
OPTION 09: IN 1 INPUT CR
OPTION 10: IN 2 INPUT = 00
    00 = TTL LOAD
    01 = TTL HI-Z
OPTION 10: IN 2 INPUT CR
OPTION 11: VCC = 00
    NOT AN OPTION
```

OPTION 12: L3 DRIVER = 02  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN  
 02 = HI CURRENT LED SEG OUT  
 03 = HI CURRENT TRI-STATE

OPTION 12: L3 DRIVER CR

OPTION 13: L2 DRIVER = 02  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN  
 02 = HI CURRENT LED SEG OUT  
 03 = HI CURRENT TRI-STATE

OPTION 13: L2 DRIVER CR

OPTION 14: L1 DRIVER = 02  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN  
 02 = HI CURRENT LED SEG OUT  
 03 = HI CURRENT TRI-STATE

OPTION 14: L1 DRIVER CR

OPTION 15: L0 DRIVER = 02  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN  
 02 = HI CURRENT LED SEG OUT  
 03 = HI CURRENT TRI-STATE

OPTION 15: L0 DRIVER CR

OPTION 16: SI INPUT = 00  
 00 = LOAD VCC  
 01 = HI-Z

OPTION 16: SI INPUT CR

OPTION 17: SO DRIVER = 02  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN  
 02 = PUSH/PULL

OPTION 17: SO DRIVER CR

OPTION 18: SK DRIVER = 02  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN  
 02 = PUSH/PULL

OPTION 18: SK DRIVER CR

OPTION 19: IN 0 INPUT = 00  
 00 = TTL LOAD  
 01 = TTL HI-Z

OPTION 19: IN 0 INPUT CR

OPTION 20: IN 3 INPUT = 00  
 00 = TTL LOAD  
 01 = TTL HI-Z

OPTION 20: IN 3 INPUT CR

OPTION 21: G0 I/O PORT = 00  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN  
 02 = STANDARD OUTPUT SMALL DRIVER  
 03 = OPEN DRAIN SMALL DRIVER

OPTION 21: G0 I/O PORT CR

OPTION 22: G1 I/O PORT = 00  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN  
 02 = STANDARD OUTPUT SMALL DRIVER  
 03 = OPEN DRAIN SMALL DRIVER

OPTION 22: G1 I/O PORT CR

OPTION 23: G2 I/O PORT = 00  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN  
 02 = STANDARD OUTPUT SMALL DRIVER  
 03 = OPEN DRAIN SMALL DRIVER

OPTION 23: G2 I/O PORT CR

OPTION 24: G3 I/O PORT = 00  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN  
 02 = STANDARD OUTPUT SMALL DRIVER  
 03 = OPEN DRAIN SMALL DRIVER

OPTION 24: G3 I/O PORT CR

OPTION 25: D3 OUTPUT = 00  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN

OPTION 25: D3 OUTPUT CR

OPTION 26: D2 OUTPUT = 00  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN

OPTION 26: D2 OUTPUT CR

OPTION 27: D1 OUTPUT = 00  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN

OPTION 27: D1 OUTPUT CR

OPTION 28: D0 OUTPUT = 00  
 00 = STANDARD OUTPUT  
 01 = OPEN DRAIN

OPTION 28: D0 OUTPUT CR

OPTION 29: COP FUNCTION = 00  
 00 = NORMAL  
 01 = MICROBUS

OPTION 29: COP FUNCTION CR

OPTION 30: COP BONDING = 00  
 00 = 28 PIN PACKAGE  
 01 = 24 AND 28 PIN PACKAGES

OPTION 30: COP BONDING CR

OPTION 31: IN INPUT LEVEL = 00  
 00 = STANDARD TTL  
 01 = HIGH TRIP POINT

OPTION 31: IN INPUT LEVEL CR

OPTION 32: G INPUT LEVEL = UNSPECIFIED  
 00 = STANDARD TTL  
 01 = HIGH TRIP POINT

OPTION 32: G INPUT LEVEL 1

OPTION 33: L INPUT LEVEL = UNSPECIFIED  
 00 = STANDARD TTL  
 01 = HIGH TRIP POINT

OPTION 33: L INPUT LEVEL 1

OPTION 34: CKO INPUT LEVEL = UNSPECIFIED  
 00 = STANDARD TTL  
 01 = HIGH TRIP POINT

OPTION 34: CKO INPUT LEVEL 0

OPTION 35: SI INPUT LEVEL = UNSPECIFIED  
 00 = STANDARD TTL  
 01 = HIGH TRIP POINT

OPTION 35: SI INPUT LEVEL 0

T> L

TRANSMITTAL FILE

RESPONSIBLE NAME/PHONE:  
 JOE COPUSER  
 (415)777-6234

COMPANY NAME/ADDRESS:  
 NATIONAL SEMICONDUCTOR  
 2900 SEMICONDUCTOR DRIVE  
 SANTA CLARA, CA 95051

DATE: JANUARY 5, 1979

FILE NUMBER: B8A7 62A0 102B

CHIP NUMBER: 420

OPTION	VALUE	OPTION	VALUE
01: GROUND	= 00	19: IN 0 INPUT	= 00
02: CKO OUTPUT	= 02	20: IN 3 INPUT	= 00
03: CKI INPUT	= 04	21: G0 I/O PORT	= 00
04: RESET INPUT	= 01	22: G1 I/O PORT	= 00
05: L7 DRIVER	= 02	23: G2 I/O PORT	= 00
06: L6 DRIVER	= 02	24: G3 I/O PORT	= 00
07: L5 DRIVER	= 02	25: D3 OUTPUT	= 00
08: L4 DRIVER	= 02	26: D2 OUTPUT	= 00
09: IN 1 INPUT	= 00	27: D1 OUTPUT	= 00
10: IN 2 INPUT	= 00	28: D0 OUTPUT	= 00
11: VCC	= 00	29: COP FUNCTION	= 00
12: L3 DRIVER	= 02	30: COP BONDING	= 00
13: L2 DRIVER	= 02	31: IN INPUT LEVEL	= 00
14: L1 DRIVER	= 02	32: G INPUT LEVEL	= 00
15: L0 DRIVER	= 02	33: L INPUT LEVEL	= 00
16: SI INPUT	= 00	34: CKO INPUT LEVEL	= 00
17: SO DRIVER	= 02	35: SI INPUT LEVEL	= 00
18: SK DRIVER	= 02		

SOURCE CHECKSUM 62A0

OBJECT CHECKSUM 102B

TRANSMIT CHECKSUM B8A7

T> CR

(Y/N,CR = YES)CR

DISK TO BE MAILED IN DRIVE FDS1: (Y/N,  
 CR = YES)? CR

CREATING FILE FDS1: MASKEK.TRN

T>

The disk is now ready to be sent to:

National Semiconductor Corp.  
 2900 Semiconductor Drive  
 Santa Clara, CA 95051

ATTN: Deborah Jacobs - D3665  
 ROM Control Customer Service  
 DISK/DISK/DISK/DISK/DISK/DISK

A mailing package, which includes a label with this information, is available from:

COPS Marketing, D3667  
 National Semiconductor  
 2900 Semiconductor Drive  
 Santa Clara, CA 95051  
 Phone: (408) 721-5883

Table 5-1. Summary of MASKTR Console Commands

Command Name	Operand Syntax	Description
<u>A</u> BORT		Aborts the creation of a Transmittal File.
<u>C</u> OMPANY		Prompts for Company Name and Address.
<u>D</u> ATE		Prompts for Date.
<u>E</u> RROR	<LPT:>	Summarizes any option conflict. LPT: sends output to the line printer.
<u>F</u> INISH		Finishes the creation of the Transmittal File.
<u>L</u> IST		Lists the Transmittal File.
<u>N</u> AME		Prompts for the Name/Phone Number of the person responsible for the program.
<u>O</u> PTION	<opt#>	Prompts for the valid options. opt# is the starting option number.
<u>P</u> RINT	<chip#>	Prints allowable options for chip specified. chip# is 410, 411, 420, 421, 422, 444, 445, 440, 441, 442, 2440, 2441 or 2442.
<u>T</u> RANSMITTAL	<filename>	Load Module is read, and entered chip number is checked against chip number in Load Module. If the chip numbers are not compatible, MASKTR aborts the Transmittal command. If they are compatible, the valid chip number is used to determine which options are valid and available. <filename> is any valid STARPLEX filename, default extension assumed is .REL.

ROM VALUES

000	00	33	5E	33	6C	2E	8D	3E	8D	91	3A	70	3E	7D	33	A8
010	7F	33	B8	7F	2E	7D	61	80	00	01	51	11	51	03	51	13
020	51	5E	49	48	00	00	00	00	00	00	00	00	00	00	00	00
030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
040	33	B8	15	5F	CC	5F	DA	5B	68	60	63	C6	00	58	21	F1
050	2C	05	5F	00	26	50	00	16	72	CA	00	58	21	EF	91	CA
060	2C	05	52	5F	48	06	25	50	23	28	16	23	38	06	48	00
070	52	55	21	CA	3A	46	CA	00	00	00	00	00	00	00	00	00
080	15	23	B9	05	23	A9	48	05	23	B9	05	04	83	00	07	8D
090	48	0E	68	8D	1D	00	52	07	95	1E	70	70	2C	70	48	3C
0A0	33	2A	40	06	4C	32	4F	5F	4D	C8	05	51	51	5F	AB	48
0B0	3F	04	04	04	04	04	04	04	C7	0E	33	3E	48	22	00	56
0C0	30	4A	07	00	56	30	4A	06	05	48	00	00	00	00	00	00
0D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
100	43	01	4B	03	4B	03	01	03	00	4B	4B	30	02	14	24	03
110	01	23	21	03	49	02	90	A0	B4	54	93	02	24	01	A0	02
120	00	CA	08	4B	4B	D8	3B	10	30	84	FC	48	80	00	C2	90
130	4A	48	0A	4A	48	42	42	48	4A	4A	CE	88	10	02	04	41
140	5F	F7	8F	39	0F	79	71	BD	F6	09	11	70	38	36	36	3F
150	F3	3F	F3	ED	01	3E	30	36	00	00	09	31	00	0E	00	08
160	00	00	20	FF	ED	ED	56	00	00	00	00	00	00	00	00	00
170	31	00	51	41	60	61	71	01	71	61	01	00	80	C8	40	83
180	1E	15	54	BF	33	2C	16	06	0F	BF	33	2C	16	06	38	15
190	33	3C	33	5F	1F	22	05	B9	4F	44	0F	05	4F	44	1E	05
1A0	4F	0E	05	3E	4F	35	50	32	4F	41	ED	6A	80	BA	33	5F
1B0	3E	35	AB	50	05	23	8F	15	23	80	05	1E	06	43	42	9F
1C0	6B	40	33	5E	3E	05	52	D0	23	3D	2A	17	05	5C	DB	D7
1D0	51	DE	23	3D	2B	68	B8	A9	A9	32	F4	6B	4D	FB	23	3D
1E0	5F	E8	2E	05	5E	E9	63	C0	A9	2D	05	3E	21	D8	3D	05
1F0	3C	32	21	22	68	18	32	2E	00	30	06	3E	AA	06	61	80

ROM VALUES

200	30	31	32	33	34	35	36	37	38	39	30	2A	2D	00	00	FF
210	00	7D	51	57	45	52	54	59	55	40	4F	50	0A	00	00	00
220	00	41	53	44	46	47	48	4A	4B	4C	3B	7F	0D	00	00	00
230	00	5A	58	43	56	42	4E	4D	2C	2E	2F	20	08	00	00	00
240	00	21	22	23	24	25	26	27	28	29	40	3A	3D	00	00	00
250	00	7D	51	57	45	52	54	59	55	49	5F	40	0A	00	00	00
260	00	41	53	44	46	47	48	4A	5B	5C	2B	7F	0D	00	00	00
270	00	5A	58	43	56	42	5E	5D	3C	3E	3F	20	08	00	00	00
280	33	A1	05	5F	C7	06	F0	07	C2	3A	11	CD	D6	33	A2	25
290	16	73	35	4E	58	CF	2F	7D	7A	33	A7	BD	5A	F4	07	BD
2A0	5A	F0	07	BD	5E	ED	33	A3	05	5C	ED	07	70	2F	7C	77
2B0	3E	05	50	48	33	A7	01	C0	F0	00	00	00	00	00	00	00
2C0	0D	00	07	C2	0F	06	1D	00	52	07	C9	1F	06	48	22	00
2D0	2B	11	32	03	D6	13	54	3D	13	53	03	52	11	51	2D	BF
2E0	33	A8	33	2C	16	06	20	42	48	00	00	00	00	00	00	00
2F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
300	0A	0D	0F	13	18	2B	38	3A	35	35	33	B8	D6	2A	DF	29
310	43	4C	DD	29	35	50	80	F5	3B	05	5E	E6	06	05	50	87
320	F5	33	B8	05	5E	D6	28	7F	38	7F	F5	2C	05	5F	E1	06
330	33	91	80	6A	C0	33	6C	48	91	E6	29	15	70	06	63	0A
340	33	01	48	33	68	39	13	DF	29	33	2C	16	06	39	05	56
350	DD	15	23	B8	05	23	A8	68	60	39	76	63	26	00	FF	01
360	E6	73	29	25	50	C9	72	29	43	4D	05	50	33	2C	07	CC
370	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
380	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
390	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
3A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
3B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
3C0	9F	5F	C6	51	68	18	61	FB	B9	3E	05	2D	06	3C	05	3D
3D0	06	2E	70	3A	03	C6	6A	CE	3B	13	E9	05	52	06	23	28
3E0	B0	23	38	B0	3A	01	60	40	C6	3F	4B	E4	00	00	00	00
3F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00



# Appendix A

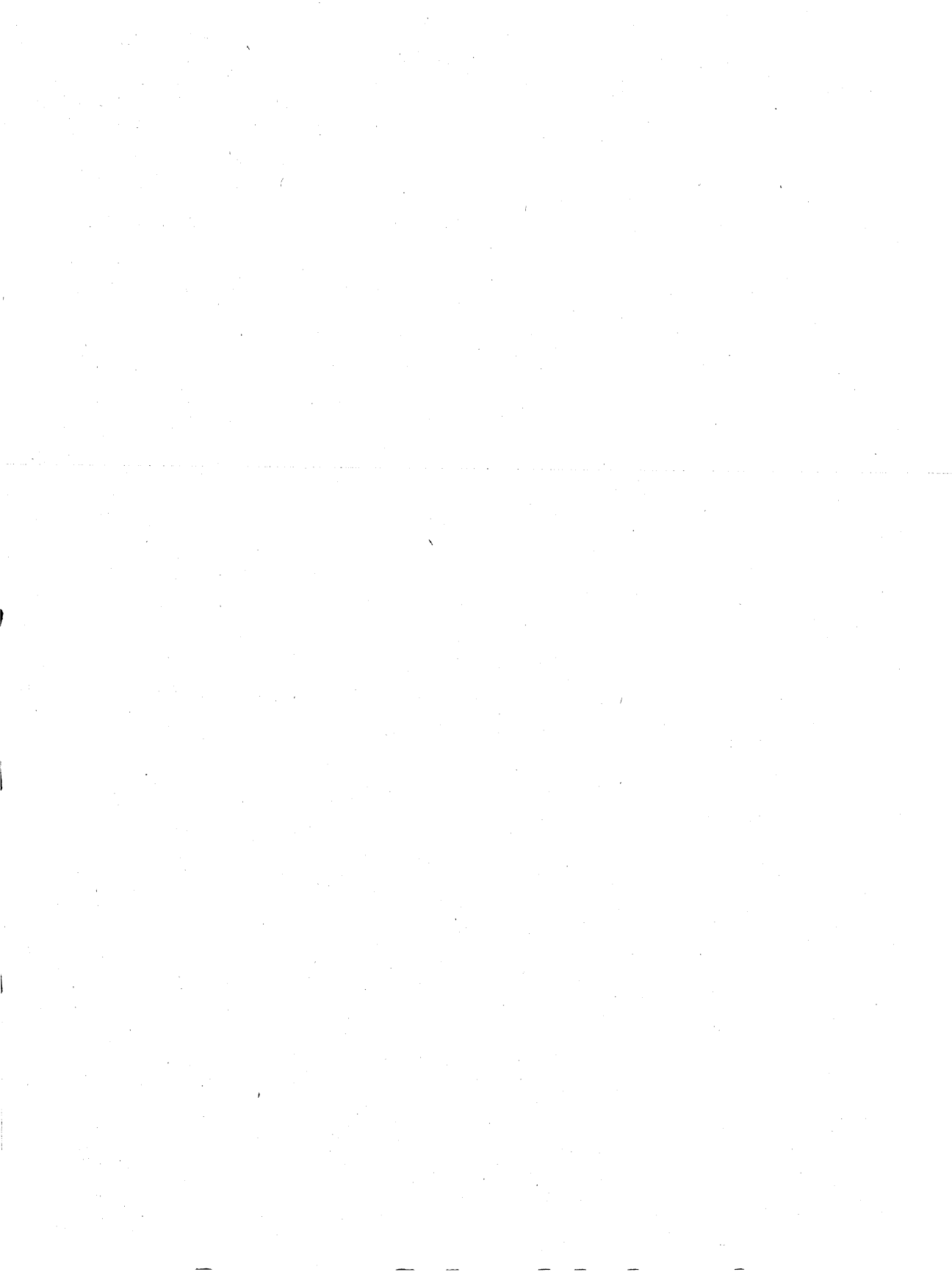
## Pinout Assignments

Emulator Board Connector	Signal Name	ISE Board J1 Pin No.	ISE Board J2 Pin No.
1	GROUND	13	
2	GROUND	25	
3	VCC	12	
4	VCC	24	
5	External Event 2	11	
6	External Event 1	23	
7	External Event 4	10	
8	External Event 3	22	
9	CLK	9	
10	SKIP	21	
11	A8	8	
12	A9	20	
13	A3	7	
14	A7	19	
15	A1	6	
16	A2	18	
17	A4	5	
18	A0	17	
19	A6	4	
20	A5	16	
21	A11	3	
22	A10	15	
23	Not used	2	
24	Not used	14	
25	Not used	1	
26	Not used		1
27	Not used		14
28	Not used		2
29	Not used		15
30	Not used		3
31	Not used		16
32	Not used		4
33	B0		17
34	B7		5
35	B2		18
36	B5		6
37	B3		19
38	B4		7
39	B6		20
40	B1		8
41	TRACE OUT (TO)		21
42	Not used		9
43	RESET*		22
44	PROM DISABLE*		10
45	-12V		23
46	-12V		11
47	VCC		24
48	VCC		12
49	GROUND		25
50	GROUND		13



**Section 9  
COPS  
Application**





# COP400 Microcontroller Family

## COPS™ Family User's Guide



# COPS Family User's Guide

## Table of Contents

Section	Description	Page
<b>Chapter 1. Introduction to COP400 Microcontrollers</b>		
1.1	Summary of COP400 Microcontroller Features .....	9-7
<b>Chapter 2. COP400 Architecture</b>		
2.1	COP420/COP421 Architecture .....	9-8
2.2	COP420/COP421 Functional Description .....	9-10
2.3	Initialization .....	9-11
2.4	COP420/COP421 Mask Programmable Options .....	9-12
2.5	COP420L/COP421L Description .....	9-15
2.6	COP420L/COP421L Mask-Programmable Options .....	9-15
2.7	COP420C Description .....	9-17
2.8	COP444L Description .....	9-18
2.9	COP402 and COP402M ROMless Part Description .....	9-18
2.10	COP404L ROMless Part Description .....	9-18
2.11	COP410L/COP411L Architecture .....	9-18
2.12	COP410L/COP411L Functional Description .....	9-20
2.13	COP410L/COP411L Mask-Programmable Options .....	9-21
<b>Chapter 3. COP400 Instruction Sets</b>		
3.1	COP420-Series/COP444L Instruction Set .....	9-23
3.2	COP420-Series/COP444L Instruction Set Description .....	9-27
3.3	COP421-Series Instruction Set Differences .....	9-34
3.4	COP410L/COP411L Instruction Set .....	9-34
3.5	COP410L/COP411L Instruction Set Differences .....	9-37
<b>Chapter 4. COP400 Programming Techniques</b>		
4.1	Program Memory Allocation .....	9-42
4.2	Data Memory Allocation and Manipulation .....	9-45
4.3	Subroutine Techniques .....	9-46
4.4	Utility Routines .....	9-47
4.5	Timing Considerations .....	9-48
4.6	BCD Arithmetic Routines .....	9-49
4.7	Simple Display Loop Routine .....	9-51
4.8	Interrupt Service Routine .....	9-53
4.9	Timekeeping Routine .....	9-53
4.10	String Search Routine .....	9-56
4.11	Programming Techniques for the COP421-Series, COP410L and 411L .....	9-57
<b>Chapter 5. COP400 I/O Techniques</b>		
5.1	Hardware Interfacing Techniques .....	9-58
5.2	Software I/O Techniques .....	9-63
5.3	Keyboard/Display Interface .....	9-64
5.4	SIO (Serial) Input/Output .....	9-75
5.5	Add-on RAM .....	9-76
5.6	IN <sub>3</sub> /IN <sub>0</sub> Inputs .....	9-77

# COPS Family User's Guide

## List of Figures

Figure	Description	Page
2.1	COP420/COP421 Block Diagram .....	9-8
2.2	COP420/COP421 Connection Diagrams .....	9-9
2.3	COP420/COP421 Pin Descriptions .....	9-9
2.4	Power-Clear Circuit .....	9-9
2.5	COP420/COP421 Clock Oscillator Configurations .....	9-13
2.6	COP420/COP421 Input/Output Configurations .....	9-14
2.7	COP420L/COP421L Oscillator Configurations .....	9-16
2.8	COP410L/COP411L Block Diagram .....	9-19
2.9	COP410L/COP411L Connection Diagrams .....	9-20
2.10	COP410L/COP411L Pin Description .....	9-20
2.11	COP410L/COP411L Oscillator Configurations .....	9-22
3.1	INIL Hardware Implementation .....	9-28
3.2	Enable Register Features — Bits EN <sub>3</sub> and EN <sub>0</sub> .....	9-33
4.1	COP420 Data Memory Map .....	9-45
4.2	Flowchart for Multiply Routine .....	9-50
4.3	Flowchart for Timekeeping Routine .....	9-54
5.1	COP420 I/O Lines .....	9-59
5.2	COP420 I/O Options .....	9-59
5.3	COP420 Standard Output Characteristics .....	9-59
5.4	COP420 I/O Interconnect Examples .....	9-60
5.5	COP420 IN Input Characteristics .....	9-60
5.6	D and G Port Characteristics .....	9-61
5.7	COP420L I/O Port Characteristics .....	9-61
5.8	COP420 SI, SO, SK Characteristics .....	9-62
5.9	COP420 OKO, OKI, RESET Characteristics .....	9-62
5.10	COP420 I/O Expansion .....	9-63
5.11	COP420 LED Display System .....	9-63
5.12	COP420 VF Display System .....	9-63
5.13	COP420 MICROBUS™ Interconnect .....	9-63
5.14	COP420 Add-On RAM .....	9-63
5.15	Display/Keyboard Interconnect .....	9-64
5.16	Flowchart for Display/Keyboard Debounce Routine .....	9-65
5.17	Display Timing Diagram .....	9-66
5.18	Display/Keyboard Interface Source Code .....	9-71
5.19	Key-Decode Routine Assembler Output Listing .....	9-74
5.20	Additional I/O Using SI and SO .....	9-75
5.21	Multi-COP420 System .....	9-75
5.22	Add-On RAM Interconnect .....	9-77

# COPS Family User's Guide

## List of Tables

Table	Description	Page
3.1	COP420/COP421 Instruction Set Table .....	9-24
3.2	COP420/COP421 Instruction Set Symbols .....	9-27
3.3	COP410L/COP411L Instruction Set Table .....	9-34
3.4	COP410L/COP411L Instruction Set Symbols .....	9-37
3.5	Alphabetical Mnemonic Index of COP420/COP421 Instructions .....	9-38
3.6	COP420/COP421 Instructions Listed by Hex Opcodes .....	9-39
3.7	Alphabetical Mnemonic Index of COP410L/COP411L Instructions .....	9-40
3.8	COP410L/COP411L Instructions Listed by Hex Opcodes .....	9-41
4.1	Page to Hexadecimal Address .....	9-42
5.1	COP400 I/O Comparison Chart .....	9-58
5.2	Seven-Segment Decode Values .....	9-68
5.3	JID Pointer Table for Display/Keyboard Routine .....	9-73

# 1 Introduction to the COP400 Microcontrollers



This manual provides information on the COP400 series of National's single-chip microcontrollers. The material contained in this manual is intended to assist the reader in understanding the internal architecture, instruction set, programming techniques, and hardware and software I/O techniques pertaining to the COP400 family of microcontroller devices.

The primary focus of this manual is the COP420 — at the time of this printing the most inclusive device, on a hardware and software level, of the COP400 family. Other members of the COP400 family are discussed primarily in terms of the less inclusive features of these other parts (i.e., the COP421, COP410L, COP411L). This approach should not result in a lack of understanding in terms of the operation and programming of these parts since they are "subset" devices of the COP420, distinguished, for the most part, by deleted hardware and software features. For further information on these other devices and on future COP400 devices the reader should consult the data sheets appropriate to particular COP400 devices.

## 1.1 Summary of COP400 Microcontroller Features

COP400 Microcontrollers are fabricated using CMOS or N-channel, silicon gate MOS technology. They are complete microcomputers containing all system timing, internal logic, ROM, RAM, and I/O necessary to implement dedicated control functions in a variety of applications. Features of the COP400 devices include an instruction set, internal architecture, and I/O scheme designed to facilitate keyboard input, display output, and efficient BCD data manipulation.

The various members of the COP400 family allow the user to specify a microcontroller best suited for use in a particular dedicated application. Specifically, COP400 devices offer a choice among single-chip parts with differing amounts of ROM, RAM, I/O capability, and number of instructions. Additionally, many parts have different versions which allow a choice of electrical characteristics while retaining the basic architecture and instruction set of the basic device. (For example, the COP420L and COP420C are available as low-power and CMOS versions, respectively, of the standard COP420 device.) Finally, each part contains a number of clock, I/O and other options,

mask-programmed into the part at the same time as the user's program; this allows even greater flexibility in matching the COP400 Microcontroller to the user's specifications, reducing the need for external interface logic.

All COP400 devices feature single-supply operation and fast, standardized, "in-house" test procedures which verify the internal logic and user program (ROM code) mask-programmed into the device. Several COP400 controllers are available in ROM-less versions for use in prototyping a COP400 system (using the COP400 Development System) or for low-volume applications.

Section 1 provides a list of COP400 devices currently available or in design, together with a summary of the basic features of each device. Refer to this manual and data sheets of particular devices for further information on these parts. Future members of the COP400 family will include more powerful hardware and software capabilities, alternative electrical specification devices (low power, CMOS versions) and peripheral devices suitable for use in many applications.

The flexible I/O configuration of COP400 Microcontrollers allows them to interface with and drive a wide range of devices using minimal external parts. Typical peripheral devices include:

1. Keyboards and displays (direct segment and digit drive possible for several devices).
2. External data memories.
3. Printers.
4. Other COPST<sup>TM</sup> devices.
5. A/D and D/A converters.
6. Power control devices (SCRs, TRIACs).
7. Mechanical actuators.
8. General purpose microprocessors (communication with host CPUs over National's MICROBUS<sup>TM</sup> for several COP400 devices).
9. Shift registers.
10. External ROM data storage devices.





Figure 2.2 shows the connection diagrams for the 28-pin COP420 and the 24-pin COP421. Figure 2.3 provides a pin description for the COP420/COP421 devices.

One should consult the COP420/COP421 data sheet for maximum ratings, DC and AC electrical characteristics for these devices.

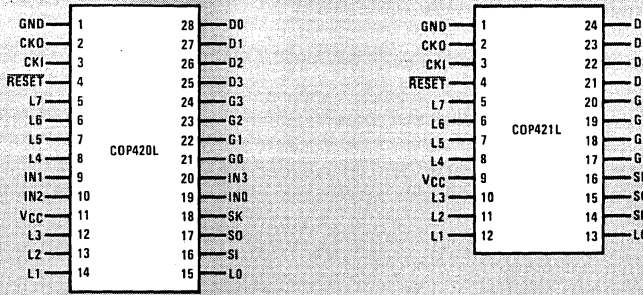


Figure 2.2. COP420/COP421 Connection Diagrams

L7-L0	8 bidirectional I/O ports with TRI-STATE <sup>®</sup>
G3-G0	4 bidirectional I/O ports
D3-D0	4 general purpose outputs
IN3-IN0	4 general purpose inputs (COP420 only)
SI	Serial input (or counter input)
SO	Serial output (or general purpose output)
SK	Logic-controlled clock (or general purpose output)
CKI	System oscillator input
CKO	System oscillator output (or general purpose input or RAM power supply)
RESET	System reset input
VCC	Power Supply
GND	Ground

Figure 2.3 COP420/COP421 Pin Description

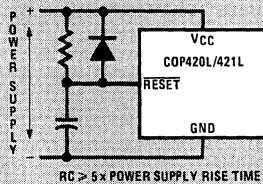


Figure 2.4. Power-Up Clear Circuit

## 2.2 COP420/COP421 Functional Description

The following text provides a functional description of the logic elements depicted in the COP420/COP421 block diagram.

### Program Memory

Program memory consists of a 1,024-byte ROM. ROM words may be program instructions, program data or ROM address pointers. Due to the special characteristics associated with the JP and JSRP instructions, ROM must often be conceived of as organized into 16 pages of 64 words (bytes) each. Also, because of the unique operations performed by the LQID and JID instructions, ROM pages must often be thought of as organized into four consecutive blocks of four ROM pages. (For further information on the paging characteristics of these instructions, see Section 4.1.)

ROM addressing is accomplished by the 10-bit P register. Its binary value selects one of the 1,024 8-bit words ( $I_7-I_0$ ) contained in ROM. The value of P is automatically incremented by 1 *prior* to the execution of the current instruction to point to the next sequential ROM location, unless the current instruction is a transfer of control instruction. In the latter case, P is loaded with the appropriate non-sequential value to implement the transfer of control operation performed by the instruction. It should be noted that P will automatically "roll-over" to point to the next page of program memory. This feature has particular significance for transfer of control instructions with paging restrictions, i.e., JP, JSRP, JID and LQID. Since P is incremented to roll-over to the next ROM page *prior* to executing these instructions, they will be treated as residing on the *next* ROM page if they reside in the last word of a ROM page. Further information is provided in Section 4.1.

Three levels of subroutine are implemented by the 10-bit subroutine save registers, SA, SB and SC, providing a last-in, first-out (LIFO) hardware subroutine stack.

ROM instruction words are fetched, decoded and executed by the Instruction Decode, Control and Skip Logic circuitry.

### Data Memory

Data memory consists of a 256-bit RAM, organized as 4 data registers of 16 4-bit digits. RAM addressing is implemented by a 6-bit B register whose upper 2 bits (Br) select 1 of 4 data registers and lower 4 bits (Bd) select 1 of 16 4-bit digits in the selected data register. While the 4-bit contents of the selected RAM digit (M) are usually loaded into or from, or exchanged with, the A register (accumulator), they may also be loaded into or from the Q latches or loaded from the L ports. RAM addressing may also be performed directly by the

LDD and XAD instructions based upon the 6-bit contents of the operand field of these instructions. The Bd register also serves as a source register for 4-bit data sent directly to the D outputs.

### Internal Logic

The 4-bit A register (accumulator) is the source and destination register for most I/O, arithmetic, logic and data memory access operations. It can also be used to load the Br and Bd portions of the B register, to load and input 4 bits of the 8-bit Q latch data, to input 4 bits of the 8-bit L I/O port data and to perform data exchanges with the SIO register.

A 4-bit adder performs the arithmetic and logic functions of the COP420, storing results in A. It also outputs a carry bit to the 1-bit C register, most often employed to indicate arithmetic overflow. The C register, in conjunction with the XAS instruction and the EN register, also serves to control the SK output. C can be outputted directly to SKL or can enable SKL to be a SYNC pulse, providing a clock each instruction cycle time. (See XAS instruction, Table 3.1, and EN register description, below.)

Four general-purpose inputs,  $IN_3-IN_0$ , are provided for the COP420:  $IN_1$ ,  $IN_2$  and  $IN_3$  may be selected, by a mask-programmable option, as Read Strobe, Chip Select and Write Strobe inputs, respectively, for use in MICROBUS™ applications.

The COP421 does not contain the  $IN_3-IN_0$  inputs and, therefore, must use the 4 bidirectional G I/O ports or 8 bidirectional L I/O ports as input pins to the device. Use of National's MICROBUS is inappropriate with the COP421.

The D register provides 4 general purpose outputs and is used as the destination register for the 4-bit contents of Bd.

The G register contents are output to 4 general-purpose bidirectional I/O ports. The COP420  $G_0$  pin may be mask-programmed as a "ready" output for MICROBUS applications.

The Q register is an internal, latched, 8-bit register, used to hold data loaded to or from M and A, as well as 8-bit program data from ROM. Its contents are output to the L I/O ports when the L drivers are enabled under program control (via an LEI instruction). The COP420 may use the MICROBUS option to write L I/O port data into Q upon the occurrence of a  $\overline{WR}$  pulse from the host CPU.

The 8 L drivers, when enabled, output the contents of latched Q data to the L I/O ports. Also, the contents of L may be read directly into A and M. As explained above, the COP420 MICROBUS option allows L I/O port data to be latched into the Q

register. L I/O ports can be directly connected to the segments of a multiplexed LED display (using the TRI-STATE™ LED Direct Drive output configuration option) with Q data being outputted to the Sa-Sg and decimal point segments of the display.

The SIO register functions as a 4-bit serial-in/serial-out shift register or as a binary counter depending on the contents of the EN register. (See EN register description, below.) Its contents can be exchanged with A, allowing it to input or output a continuous serial data stream. SIO may also be used to provide additional parallel I/O when used as a shift register with its input or output connected to external serial-in/parallel-out shift registers.

The 10-bit time base counter divides the instruction cycle frequency by 1,024, providing a pulse upon overflow. The COP420 SKT instruction tests for the occurrence of this pulse, allowing the programmer to rely on this internal time-base rather than external inputs (e.g., 50/60 Hz signals) to implement "real-time" routines.

The EN register is an internal 4-bit register loaded under program control by the LEI instruction. The state of each bit of this register selects or deselects the particular feature associated with each bit of the EN register (EN<sub>3</sub>-EN<sub>0</sub>).

1. The least significant bit of the enable register, EN<sub>0</sub>, selects the SIO register as either a 4-bit shift register or a 4-bit binary counter. With EN<sub>0</sub> set, SIO is an asynchronous binary counter, decrementing its value by one upon each low-going pulse ("1" to "0") occurring on the SI input (count-down counter). Each pulse must be at least two instruction cycles wide. SK outputs the value of C upon execution of XAS and remains latched until the execution of another XAS instruction. The SO output is equal to the value of EN<sub>3</sub>. With EN<sub>0</sub> reset, SIO is a serial shift register shifting left each instruction cycle time. The data present at SI goes into the least significant bit of SIO. SO can be enabled to output the most significant bit of SIO each cycle time. The SK output becomes a logic-controlled clock, providing a SYNC signal each instruction time. It will start outputting a SYNC pulse upon the execution of an XAS instruction with C = 1, stopping upon the execution of a subsequent XAS with C = 0.
2. With EN<sub>1</sub> set, the COP420 IN<sub>1</sub> input is enabled as an interrupt input. Immediately following an interrupt, EN<sub>1</sub> is reset to disable further interrupts. Note that this interrupt feature associated with IN<sub>1</sub> is unavailable on the COP421 since it lacks the IN inputs. Bit 1 (EN<sub>1</sub>)

of the Enable Register is, therefore, a "don't care" bit for the COP421: setting or resetting this bit via an LEI instruction will have no effect on the operation of the COP421. (For further information on the procedure and protocol of this COP420 interrupt feature, see Section 3.2, LEI instruction description.)

3. With EN<sub>2</sub> set, the L drivers are enabled to output the data in Q to the L I/O ports. Resetting EN<sub>2</sub> disables the L drivers, placing the L I/O ports in a high-impedance input state. If the COP420 MICROBUS™ option is being used, EN<sub>2</sub> does not affect the L drivers.
4. EN<sub>3</sub>, in conjunction with EN<sub>0</sub>, affects the SO output. With EN<sub>0</sub> set (binary counter option selected), SO will output the value loaded into EN<sub>3</sub>. With EN<sub>0</sub> reset (serial shift register option selected), setting EN<sub>3</sub> enables SO as the output of the SIO shift register, outputting serial shifted data each instruction time. Resetting EN<sub>3</sub> with the serial shift register option selected disables SO as the shift register output: data continues to be shifted through SIO and can be exchanged with A via an XAS instruction but SO remains reset to "0." Table 2.1 provides a summary of the options and features associated with EN<sub>3</sub> and EN<sub>0</sub>.

### 2.3 Initialization

Upon initialization of the COP420/COP421 as described below, the P register is cleared to 0 (ROM address 0) and the A, B, C, D, EN, and G registers are cleared. The IN<sub>0</sub> and IN<sub>3</sub> latches are not cleared. The SK output is enabled as a SYNC output, providing a pulse each instruction cycle time. *Data memory (RAM) can only be cleared by the user's program. The first instruction at address 0 must be a CLRA.*

The Reset Logic, internal to the COP420/COP421, will initialize (clear) the device upon power-up if the power supply rise time is less than 1 ms and greater than 1 μs. If the power supply rise time is greater than 1 ms, the user must provide an external RC network and diode to the  $\overline{\text{RESET}}$  pin as shown in Figure 2.4 below. The  $\overline{\text{RESET}}$  pin is configured as a Schmitt trigger input. If not used, it should be connected to V<sub>CC</sub>. Initialization will occur whenever a logic "0" is applied to the  $\overline{\text{RESET}}$  input, provided it stays low for at least three instruction cycle times. In order to reset the Time Base Counter, a  $\overline{\text{RESET}}$  pulse ten instruction cycle times wide must be applied; note that the counter will overflow and generate an output pulse.

## 2.4 COP420/COP421 Mask Programmable Options

To allow even greater flexibility in specifying a COP400 device appropriate to the user's application, all COP400 microcontrollers have specific clock configuration, I/O and other mask-programmable options associated with them. These options are masked into the part simultaneously with the masking of the user's program in ROM and have been chosen to offer the user a wide range of options which encompasses design options most frequently employed in dedicated, small system applications.

The following text summarizes the COP420/COP421 options according to the various functions (oscillator, I/O, etc.) with which they are associated.

### Clock Oscillator Options

There are four basic COP420/COP421 clock oscillator configurations available as shown by Figure 2.5 (a-d):

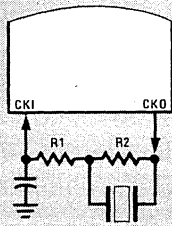
- a. Crystal Controlled Oscillator. CKI and CKO are connected to an external crystal. The instruction cycle time equals the crystal frequency (4 MHz maximum) divided by 16 (optional by 8).
- b. External Oscillator. CKI is configured as a TTL compatible input accepting an external clock signal. The external frequency (4 MHz maximum) is divided by 16 (optional by 8) to derive the instruction cycle time. CKO is now available to be used as the RAM power supply ( $V_R$ ) pin, as a general purpose input, or as a synchronizing input.
- c. RC Controlled Oscillator. CKI is configured as a single-pin RC controlled Schmitt trigger oscillator. The instruction cycle equals the oscillation frequency divided by 4. CKO is available for non-timing functions as in b above.
- d. Externally Synchronized Oscillator. Intended for use in multi-COP systems, CKO is programmed to function as an input connected to the SK output of another COP420/COP421 with CKI connected as shown. In this configuration, the SK output connected to CKO must provide a SYNC (instruction cycle) signal to CKO, thereby allowing synchronous data transfer between the COPs using only the SI and SO serial I/O pins in conjunction with the XAS instruction. Note that on power-up SK is automatically enabled as a SYNC output. (See Initialization, above.)

The lower portion of Figure 2.5 provides component values for several instruction cycle times and crystal values associated with the RC controlled and Crystal Oscillator options, respectively.

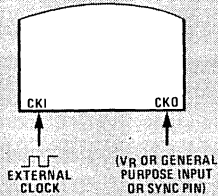
### CKO Non-Timing Options

In a crystal controlled or multi-COP oscillator system, CKO is used as an output to the crystal network. In the other two configurations (external clock or RC controlled oscillator), CKO may be mask-programmed to perform one of two available options. Specifically, CKO may be mask-programmed as a general purpose input, read into bit 1 of the accumulator ( $A_2$ ) upon the execution of an INIL instruction.

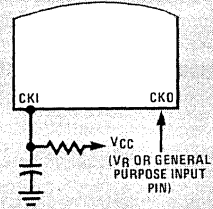
As another option (for both the COP420 and COP421), CKO can be a RAM power supply pin ( $V_R$ ), allowing its connection to a standby/backup power supply to maintain the integrity of RAM data with minimum power drain when the main supply is inoperative or shut down to conserve power. Use of this options should include external circuitry to detect loss of  $V_{CC}$  power and force RESET low before  $V_{CC}$  drops below spec.



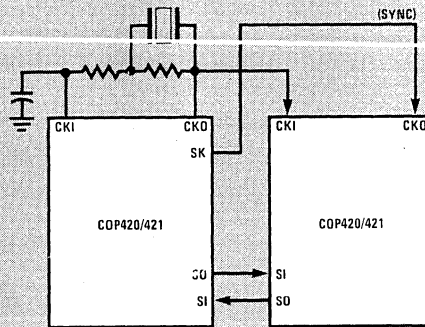
a. Crystal Oscillator



b. External Oscillator



c. RC Controlled Oscillator



d. Externally Synchronized Oscillator

Crystal Oscillator

Crystal Value	Component Values		
	R1	R2	C
4 MHz	1k	1M	27 pF
3.58 MHz	1k	1M	27 pF
2.09 MHz	1k	1M	56 pF

RC Controlled Oscillator

R (k $\Omega$ )	C (pF)	Instruction Cycle Time
		( $\mu$ s)
12	100	5 $\pm$ 20%
6.8	220	5.3 $\pm$ 23%
8.2	300	8 $\pm$ 29%
22	100	8.6 $\pm$ 16%

Figure 2.5 COP420/COP421 Oscillator Configurations



### MICROBUS™ Option

The COP420 has an option which allows it to be used as a peripheral microprocessor device, inputting and outputting data from and to a host microprocessor ( $\mu P$ ).  $IN_1$ ,  $IN_2$ , and  $IN_3$  general purpose inputs become MICROBUS compatible read-strobe, chip-select, and write-strobe lines, respectively.  $IN_1$  becomes  $\overline{RD}$  — a logic "0" on this input will cause Q latch data to be enabled to the L ports for input to the  $\mu P$ .  $IN_2$  becomes  $\overline{CS}$  — a logic "0" on this line selects the COP420 as the  $\mu P$  peripheral device by enabling the operation of the  $\overline{RD}$  and  $\overline{WR}$  lines and allows for the selection of one of several peripheral components.  $IN_3$  becomes  $\overline{WR}$  — a logic "0" on this line will write bus data from the L ports to the Q latches for input to the COP420.  $G_0$  becomes a "ready" output, reset by a write pulse from the  $\mu P$  on the  $\overline{WR}$  line, providing the "handshaking" capability necessary for asynchronous data transfer between the host CPU and the COP420.

This option has been designed for compatibility with National's MICROBUS — a standard interconnect system for 8-bit parallel data transfer between MOS/LSI CPUs and interfacing devices. (See MICROBUS™, National Publication.) The functioning and timing relationships between the COP420 signal lines affected by this option are as specified for the MICROBUS interface. Connection of the COP420 to the MICROBUS is shown in Figure 5.13.

### I/O Options

COP420/421 outputs have the following optional configurations, illustrated in Figure 2.6:

- a. **Standard** — an enhancement mode device to ground in conjunction with a depletion-mode device to  $V_{CC}$ , compatible with TTL and CMOS input requirements. Available on SO, SK, and all D and G outputs.
- b. **Open-Drain** — an enhancement-mode device to ground only, allowing external pull-up as required by the user's application. Available on SO, SK, and all D and G outputs.
- c. **Push-Pull** — An enhancement-mode device to ground in conjunction with a depletion-mode device paralleled by an enhancement-mode device to  $V_{CC}$ . This configuration has been provided to allow for fast rise and fall times when driving capacitive loads. Available on SO and SK outputs only.
- d. **Standard L** — same as a., but may be disabled. Available on L outputs only.
- e. **Open Drain L** — same as b., but may be disabled. Available on L outputs only.
- f. **LED Direct Drive** — an enhancement-mode device to ground and to  $V_{CC}$ , meeting the typical current sourcing requirements of the segments of an LED display. the sourcing device is clamped to limit current flow. These devices may be turned off under program control (See Functional Description, EN Register), placing the outputs in a high-impedance state to provide required LED segment blanking for a multiplexed display.

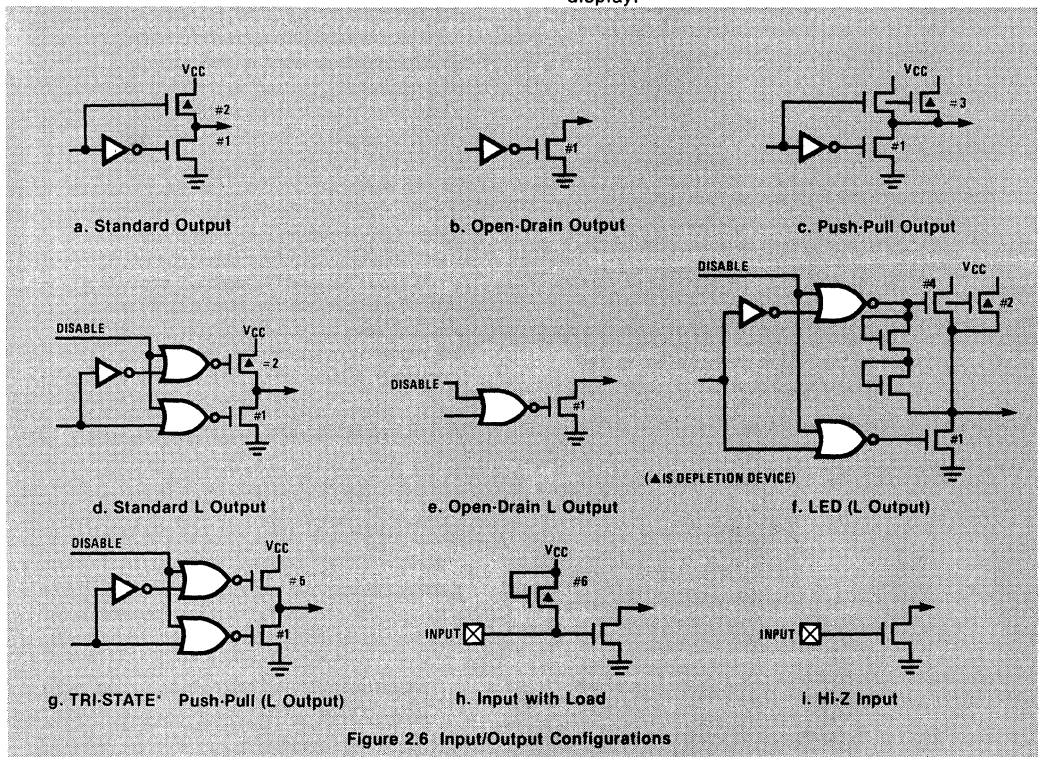


Figure 2.6 Input/Output Configurations

g. **TRI-STATE® Push-Pull** — an enhancement-mode device to ground and  $V_{CC}$ . These outputs are TRI-STATE outputs, allowing for connection of these outputs to a data bus shared by other bus drivers.

COP420/COP421 inputs have the following optional configurations:

- h. An on-chip depletion load device to  $V_{CC}$ .
- i. A Hi-Z input which must be driven to a "1" or "0" by external components.

The above input and output configurations share common enhancement-mode and depletion-mode devices. Specifically, all configurations use one or more of six devices (numbered 1-6, respectively).

The SO, SK outputs can be configured as shown in a., b., or c. The D and G outputs can be configured as shown in a. or b. Note that when inputting data to the G ports, the G outputs should be set to "1." The L outputs can be configured as in d., e., f. or g.

An important point to remember if using configuration d. or f. with the L drivers is that even when the L drivers are disabled, the depletion load device will source a small amount of current; however, when the L lines are used as inputs, the disabled depletion device can *not* be relied on to source sufficient current to pull an input to logic "1".

All of the L driver options are TRI-STATE®-able. Therefore, the L drivers have TRI-STATE-able Standard and Open-Drain output options as well as the TRI-STATE LED Direct Drive and Push-Pull output options. Since the device to  $V_{CC}$  in the Standard output configuration is a depletion-mode device, it will source up to 0.125mA when this output is "turned off" in the TRI-STATE mode. This is not a worst case input for a logic "1" level on these inputs and will not be sufficient for an input level without previously enabling Q to L with  $(Q) = FF_{16}$ .

#### Bonding Option

The COP421 is a bonding option of the COP420: if the COP420 is bonded as a 24-pin device (without the 4 IN inputs), it becomes the COP421. Note that since it lacks the IN inputs, use of the COP421 bonding option precludes use of the IN input options; the MICROBUS™ option which would otherwise affect  $IN_3$ - $IN_1$  and  $G_0$ ; use of the  $IN_1$  hardware interrupt pin and the use of the  $IL_3$  and  $IL_0$  latches associated with the  $IN_3$  and  $IN_0$  pins. All other options are available. The COP421 is pin-compatible with the COP410L.

## 2.5 COP420L/COP421L Description

The COP420L/COP421L are low power versions of the COP420/COP421 containing the *same* internal logic elements and instruction set as the COP420/COP421, with *electrical* characteristics which are similar to the COP410L. The major differences between the COP420L/COP421L and COP420/COP421 are the following:

- Wider operating voltage range of 4.5 to 9.5V optionally available.
- Operating supply current less than 8mA @  $V_{CC} = 5V$ .
- Minimum instruction cycle time of 15 $\mu$ s.
- Divide-by-32 crystal clock option (2MHz XTAL divided by 32 = 15 $\mu$ s instruction cycle time).
- D and G outputs have direct LED digit drive option (sink 30mA).
- Other outputs will drive 1 LS11L or 2 LPTTL loads ( $I_{OL} = 360\mu A$  at 0.4V;  $I_{OH} = 40\mu A$  at 2.4V).
- No MICROBUS™ option available.

The COP421L is simply a COP420L packaged in a 24-pin dual-in-line package. As a result, the IN inputs are not available on the COP421L, so that the COP421L is pin-compatible with the COP410L.

For further information, see the COP420L/COP421L data sheet.

## 2.6 COP420L/COP421L Mask Programmable Options

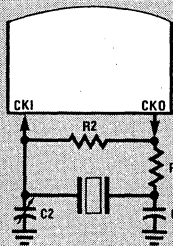
Since the COP420L/COP421L are frequently used in battery-operated and/or hand-held consumer-type products, an even greater array of system-cost-reducing options is available. The following text summarizes these options.

#### Clock Oscillator Options

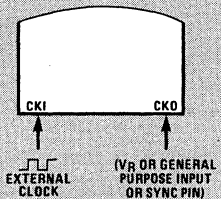
There are four basic COP420L/COP421L clock oscillator configurations available as shown in Figure 2.8 (a-d):

- a. Crystal/Resonator Controlled Oscillator. CKI and CKO are connected to an external crystal or ceramic resonator. The instruction cycle time equals the crystal/resonator frequency (2.097 MHz maximum) divided by 32 (optional by 16 or 8).
- b. External Oscillator. CKI is configured as a CMOS compatible input accepting an external clock signal. The external frequency (2MHz maximum) is divided by 32 (optional by 16, 8 or 4) to derive the instruction cycle time. CKO is now available to be used as the RAM power supply ( $V_R$ ) pin, as a COP420L general purpose input, or as a synchronizing input.

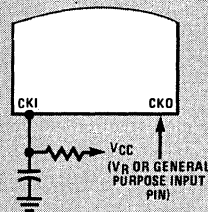




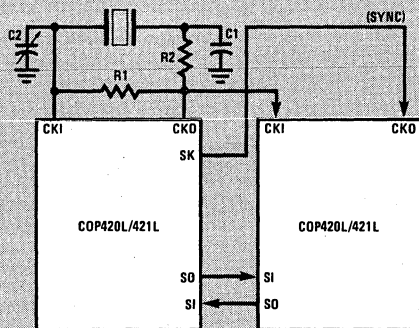
a. Crystal Oscillator



b. External Oscillator



c. RC Controlled Oscillator



d. Externally Synchronized Oscillator

Crystal Oscillator

Crystal Value	Component Values			
	R1	R2	C1	C2
455 kHz (Resonator)	16k	1M	80 pF	80 pF
2.09 MHz	1k	1M	56 pF	6-36 pF

RC Controlled Oscillator

R (kΩ)	C (pF)	Instruction Cycle Time (μs)
51	100	19 ± 15%
82	56	19 ± 15%

Figure 2.7 COP420L/COP421L Oscillator Configurations

- c. RC Controlled Oscillator. CKI is configured as a single-pin RC controlled Schmitt trigger oscillator. The instruction cycle equals the oscillation frequency divided by 4. CKO is available for non-timing functions as in b above.
- d. Externally Synchronized Oscillator. Intended for use in multi-COP systems, CKO is programmed to function as an input connected to the SK output of another COP420L/COP421L with CKI connected as shown. In this configuration, the SK output connected to CKO must provide a SYNC (instruction cycle) signal to CKO, thereby allowing synchronous data transfer between the COPs using only the SI and SO serial I/O pins in conjunction with the XAS instruction. Note that on power-up SK is automatically enabled as a SYNC output.

The lower portion of Figure 2.7 provides component values for several instruction cycle times and crystal values associated with the RC controlled and crystal controlled oscillator options, respectively.

#### CKO Non-Timing Options

In a crystal controlled or multi-COP oscillator system, CKO is used as an output to the crystal network. In the other two configurations (external clock or RC controlled oscillator), CKO may be mask-programmed to perform one of two available options. Specifically, CKO may be mask-programmed as a general purpose COP420L input, read into bit 1 of the accumulator ( $A_2$ ) upon the execution of an INIL instruction.

As another option (for both the COP420L and COP421L), CKO can be a RAM power supply pin ( $V_R$ ), allowing its connection to a standby/backup power supply to maintain the integrity of RAM data with minimum power drain when the main supply is inoperative or shut down to conserve power.

#### I/O Options

While the COP420L/COP421L has capabilities to directly drive LED displays through increased voltage and current specs, the circuit configurations are identical to those of the COP420 in Figure 2.6. Increased current sink and source values are a result of changing device sizes (within the bounds of the same circuit configuration). When emulating the COP420L with the COP402, one might use the typical values of the 402 as worst case COP420L drive parameters. An alternative is the use of the COP404L to emulate the drive of the COP420L.

For detailed electrical characteristics, refer to the COP420L/COP421L data sheet.

The SO and SK outputs can be configured as shown in Figure 2.6, a, b, or c. The D and G outputs can be configured as shown in a or b. Note that when inputting data to the G ports, the G outputs should be set to "1." The L outputs can be configured as shown in d, e, f, or g.

An important point to remember is that *all* of the L driver options are TRI-STATE<sup>®</sup>-able. Therefore, the L drivers have TRI-STATE-able Standard and Open-Drain output options as well as the TRI-STATE LED Direct Drive and Push-Pull output options. Since the device to  $V_{CC}$  in the Standard output configuration is a depletion-mode device, it will source up to 0.125mA when this output is "turned off" in the TRI-STATE mode, which is insufficient to guarantee a logic "1" input level.

#### Bonding Option

The COP421L is a bonding option of the COP420L: if the COP420L is bonded as a 24-pin device (without the 4 IN inputs), it becomes the COP421L. The COP421L is pin-compatible with the COP410L.

## 2.7 COP420C Description

The COP420C is a CMOS version of the COP420. It differs from the COP420 primarily in electrical specifications; however, it also features a dual clock mode option for operation at low speed (typically 244 $\mu$ s instruction cycle time) with low power consumption (25 $\mu$ A with  $V_{CC} = 2.4$ V) or high speed (15 $\mu$ s instruction cycle time) when necessary to perform internal data computations at a faster rate. The COP420C has the same output drive characteristics as the COP420 (TTL/CMOS compatible) and retains the MICROBIUSTM option. The following are the major differences between the COP420C and the COP420:

- Operating voltage of 2.4V to 6.0V.
- Low power consumption at 244 $\mu$ s instruction cycle time (inexpensive 32kHz XTAL  $\pm$  8) = 25 $\mu$ A at  $V_{CC} = 2.4$ V.
- Dual clock mode option allowing operation at 16 $\mu$ s instruction cycle time (using external RC network) for internal data computation operations.
- "Fast" clock mode entered under program control.

For further information, see the COP420C data sheet.

## 2.8 COP444L/COP445L Description

The COP444L/COP445L are expanded-memory versions of the COP420L containing the same internal logic elements and instruction set as the COP420 and COP420L, but with twice the amounts of ROM and RAM. The major differences between the COP444L/COP445L and the COP420L/COP421L are the following:

- Operating supply current less than 11 mA at  $V_{CC} = 5V$ .
- 2048 × 8 ROM.
- 128 × 4 RAM.

The COP445L is simply a COP444L in a 24-pin dual-in-line package. As a result, the IN inputs are not available on the COP445L, so that the COP445L is pin-compatible with the COP421L and COP410L.

These devices are emulated using the COP404L.

For further information, see the COP444L/445L and/or COP404L data sheets.

## 2.9 COP402 and COP402M ROM-Less Parts Description

The COP402 and COP402M are ROM-less versions of the COP420. They are packaged in 40-pin packages and are available for prototyping a COP420 system using the COP400 Development System (PDS) or, in quantity, for small volume applications using external ROM.

The COP402 has been mask programmed with options suitable for use as a general controller. COP402 inputs have load devices to  $V_{CC}$ , the various outputs have the fullest drive capability

associated with them (L outputs = LED direct drive; G and D outputs = standard; SO, SK outputs = push-pull). The COP402 has been programmed for use with an external crystal network, using CKI and CKO, with an instruction cycle time equal to the crystal frequency divided by 16.

The COP402M is the MICROBUS™ compatible version of the COP402. It features the same options as the COP402 with the single exception that the MICROBUS option has been selected. It is, of course, intended for use in prototyping systems or small volume applications which use the microcontroller as a CPU peripheral component, with communication over National's MICROBUS.

## 2.10 COP404L ROM-Less Part Description

The COP404L is a ROM-less version of the COP444L. It is packaged in a 40-pin package and may be used to prototype all low-power COP400 devices (COP411L, COP410L, COP420L, COP421L, COP444L).

## 2.11 COP410L/COP411L Architecture

Figure 2.9 provides a block diagram of the COP410L/COP411L. As with the COP420/COP421 block diagram, it depicts the internal logic and interconnects of the device in simplified form. Note that the COP410L is functionally a subset of the 24-pin COP421L. As with the COP421L, it lacks the COP420L IN inputs and the internal IL latches associated with two of these deleted input pins. These and other architectural differences are discussed in the Functional Description, below.

Figure 2.10 shows the Connection Diagrams for the 24-pin COP410L and the 20-pin COP411L. Figure 2.11 provides a pin description for the COP410L/COP411L devices.

See data sheet for the electrical specifications of the COP410L/COP411L, showing maximum ratings plus DC and AC characteristics for these devices.

The COP401L is available for final program verification for a COP410L/COP411L application.

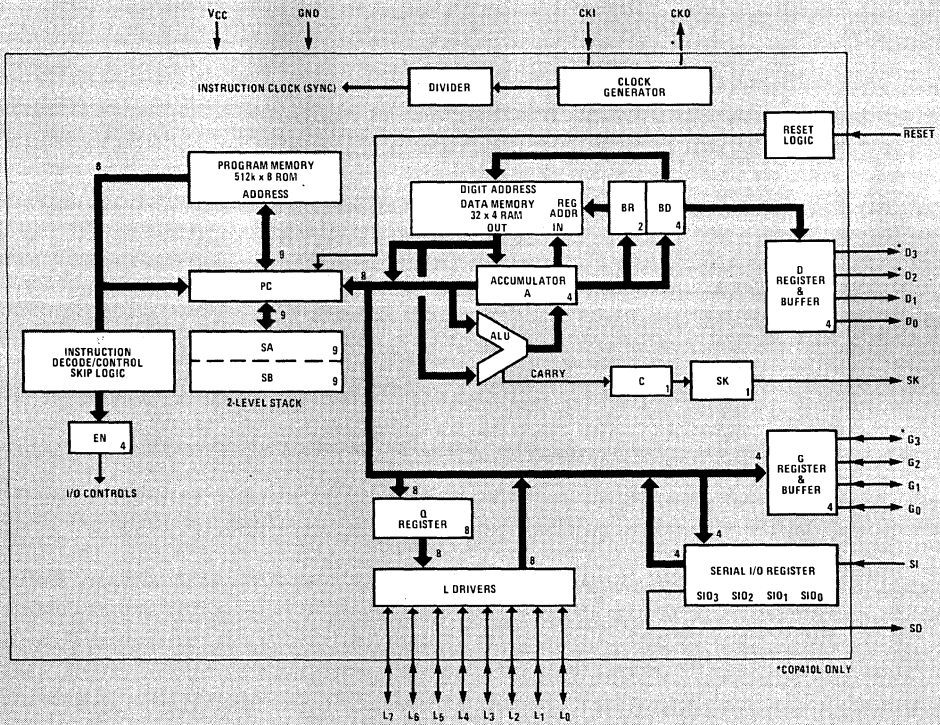


Figure 2.8 COP410L/COP411L Block Diagram

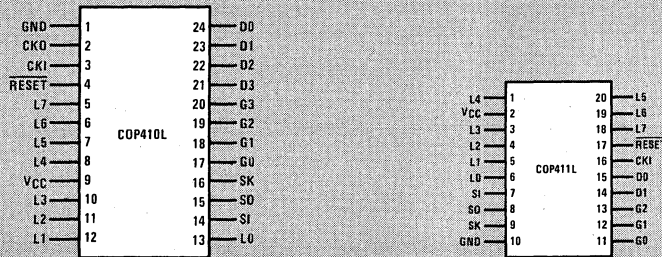


Figure 2.9 COP410L/COP411L Connection Diagrams

L <sub>7</sub> -L <sub>0</sub>	8 bidirectional I/O ports with TRI-STATE®	CKI	System oscillator input
G <sub>3</sub> -G <sub>0</sub>	4 bidirectional I/O ports	CKO	System oscillator output (or RAM power supply)
D <sub>3</sub> -D <sub>0</sub>	4 general purpose outputs	RESET	System reset input
SI	Serial input (or counter input)	V <sub>CC</sub>	Power supply
SO	Serial output (or general purpose output)	GND	Ground
SK	Logic-controlled clock (or general purpose output)		

Figure 2.10 COP410L/COP411L Pin Description

## 2.12 COP410L/COP411L Functional Description

The following text provides a functional description of the differences which exist between the internal architecture of the COP420, covered in detail in Section 2.2, and that of the COP410L and COP411L. Consequently, for information on logic elements not discussed below which appear in Figure 2.9, COP410L/COP411L Block Diagram, refer to Section 2.2. Where appropriate, differences between the COP410L and its smaller version, the COP411L, are noted in the following text.

### Program Memory

Program memory consists of a 512-byte ROM. The same paging characteristics apply to the COP410L/COP411L when allocating program memory instruction code as those which apply to the COP420 (see Section 4.1) except that ROM consists of 8 (0-7) pages of 64 (0-63) words each.

ROM addressing is accomplished by a 9-bit P register. The auto increment-before-execution and page-rollover features of the COP420 apply to the COP410L/COP411L.

Since the COP410L/COP411L have 2 9-bit subroutine-save registers, SA and SB, subroutine nesting is allowable to two levels (only one level when executing a LQID instruction since this instruction pushes the stack).

### Data Memory

Data memory consists of a 128-bit RAM organized as 4 (0-3) data registers of 8 4-bit digits. Digit addressing is valid only for digits 0, 9-15 in a particular register. (The COP410L/COP411L will, however, treat digit addresses of 1-7 as valid digit values of 9-15, respectively.) As with the COP420, RAM addressing is accomplished by a 6-bit B register whose upper 2 bits (Br) select 1 of 4 data registers and lower 3 bits (Bd) select 1 of 8 4-bit digits.

A direct access to data memory, without using the B register, is only permissible with respect to M(3, 15) by using an XAD 3, 15 instruction. All other XAD and all LDD instructions have been deleted from the COP410L/COP411L instruction set. Consequently, all other RAM locations must be accessed by loading the B register with the address of data memory to be accessed.

As with the COP420, Bd also may be used as a source register to output its 4-bit contents directly to the D outputs via an OBD instruction.

The Q register functions in a similar manner as the COP420 Q register with the following exceptions:

1. Its contents must be read with the INL instruction, since the CQMA instruction has been deleted.
2. It cannot be loaded with the contents of the L I/O ports since this function is associated with the deleted MICROBUS™ option.

The COP410L/COP411L does not contain the COP420 internal divide-by-1024 time-base counter; hence, the SKT instruction has been deleted. "Real-time" program counters must, therefore, rely on an external time-base input (e.g., 50/60 Hz square wave) to derive a program "clock" for such applications, rather than on the COP410L/COP411L instruction cycle clock itself.

Bit 1 of the EN register (EN<sub>1</sub>) is a "don't care" bit, as explained above, due to the lack of a COP410L/COP411L IN<sub>1</sub> input. (The COP420 uses the EN<sub>1</sub> bit to enable IN<sub>1</sub> as an interrupt signal.)

The CASC, ADT and OGI instructions have been deleted. See Section 3.4 for hints on performing these functions.

### 2.13 COP410L/COP411L Mask Programmable Options

The following text describes the differences which exist between the COP420L mask programmable options and those which are available for the COP410L and COP411L devices.

Available clock oscillator configurations are as follows:

- a. Ceramic Resonator Controlled Oscillator. CKI and CKO are connected to an external ceramic resonator. The instruction cycle time equals the resonator frequency (500kHz maximum) divided by 8. This configuration and its associated options are not available on the 20-pin COP411L since it lacks the CKO pin.
- b. External Oscillator. CKI is configured as a Schmitt trigger input (not TTL compatible), accepting an external clock signal. The external frequency (500kHz maximum) is divided by 8 to derive the instruction cycle time. This option applies to both the COP410L and the COP411L. For the COP410L, moreover, this configuration allows CKO to be used for a RAM power supply (V<sub>R</sub>).
- c. RC Controlled Oscillator. CKI is configured as a single pin RC controlled Schmitt trigger oscillator. The instruction cycle equals the oscillator (RC time-constant) frequency divided by 4.
- d. Externally Synchronized Oscillator. CKO is configured as a synchronizing input from the SK

output of another COP400 device. CKI is an external oscillator (divide by 8).

The lower portion of Figure 2.11 provides component values associated with the RC controlled oscillator option.

#### COP410L CKO Non-Timing Options

In the COP410L resonator controlled configuration, CKO is used as an output to the resonator network. In the other two configurations (external clock and RC controlled), CKO may be mask-programmed as a RAM power supply pin (V<sub>R</sub>), allowing its connection to a standby battery backup power supply to maintain the integrity of RAM data with minimum power drain when the main supply is inoperative or shut down to conserve power.

#### COP410L/COP411L I/O Options

COP410L/COP411L *inputs* and *outputs* have the same optional configurations as the COP420L/COP421L; see Section 2.7.

The input and output configurations share common enhancement-mode and depletion-mode devices. For detailed electrical characteristics on these devices, refer to the COP410L and COP421L data sheets.

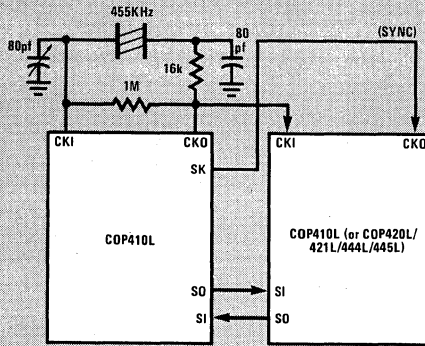
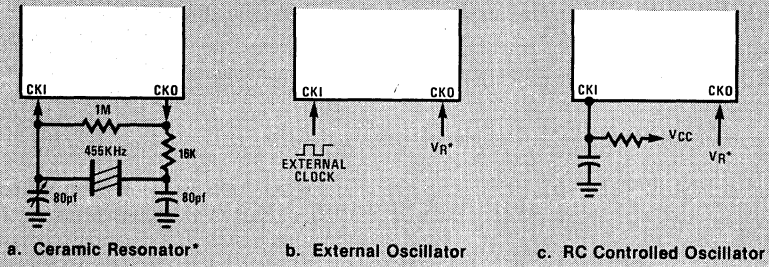
The SO and SK outputs can be configured as shown in Figure 2.6, a, b, or c. The D and G outputs can be configured as shown in a or b. Note that when inputting data to the G ports, the G outputs should be set to "1." The L outputs can be configured as shown in d, e, f, or g.

An important point to remember is that *all* of the L driver options are TRI-STATE® -able. Therefore, the L drivers have TRI-STATE-able Standard and Open-Drain output options as well as the TRI-STATE LED Direct Drive and Push-Pull output options. Since the device to V<sub>CC</sub> in the Standard output configuration is a depletion-mode device, it will source up to 0.125mA when this output is "turned off" in the TRI-STATE mode, which is insufficient to guarantee a logic "1" input level.

#### Bonding Option

The COP411L is a bonding option of the COP410L: if the COP410L is bonded as a 20-pin device (without CKO, D<sub>2</sub>, D<sub>3</sub>, and G<sub>3</sub>), it becomes the COP411L. Use of output options associated with these deleted pins are, of course, precluded. All other COP410L options are available.





\*COP410L only.

d. Externally Synchronized Oscillator\*

RC Controlled Oscillator

R (k $\Omega$ )	C (pF)	Instruction Cycle Time ( $\mu$ s)
51	100	19 $\pm$ 15%
82	56	19 $\pm$ 15%

Figure 2.11 COP410L/COP411L Oscillator Configurations

# 3 COP400 Instruction Sets



This chapter provides information on the instruction sets of the COP400 microcontrollers. As with the architecture of the different devices in the COP400 family, the instruction sets of the various devices allow the user to choose among several devices to provide only as much software capability as is needed for a particular application. Specifically, the instruction sets of the various devices are, generally, subsets of the most inclusive instruction set of the COP440. This chapter will discuss the COP420-series (includes COP421, COP421L, COP421C), COP444L, COP410L, and COP411L, respectively. Users of the COP440 should refer to the COP440 data sheet (when the device becomes available) for information on the additional instructions associated with the COP440 instruction set.

This chapter primarily provides information on the machine operations associated with the instruction set of COP400 devices. However, where appropriate, short examples indicating typical usage of particular instructions are provided. For a detailed treatment on using COP400 instructions to write COP400 assembly language programs, see Chapter 4 of this manual.

## 3.1 COP420-Series/COP444L Instruction Set

Table 3.1 provides the mnemonic, operand, machine code, data flow, skip conditions and description associated with each instruction in the COP420-series/COP444L instruction set. As indicated, an asterisk in the description column signifies a double-byte instruction. Also, notes are provided following this table which describe or refer to additional information relevant to particular instructions. As indicated by Note 3, the INI and INIL instructions are not included in the COP421 instruction set, due to its lack of IN inputs and the IL<sub>3</sub> and IL<sub>0</sub> latches associated with two of the IN inputs (IN<sub>3</sub> and IN<sub>0</sub>, respectively).

Note that the COP420-series/COP444L set, as with all COP400 instruction sets, is divided into the following categories: Arithmetic Operations, Input/Output Instructions, Transfer of Control Instructions, Memory Reference Instructions, Register Reference Instructions, and Test Instructions.



Table 3.1 COP420 Series/COP444L Instruction Set

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>ARITHMETIC INSTRUCTIONS</b>						
ASC		30	<u>0 0 1 1</u>   <u>0 0 0 0</u>	$A + C + RAM(B) \rightarrow A$ Carry $\rightarrow C$	Carry	Add with Carry, Skip on Carry
ADD		31	<u>0 0 1 1</u>   <u>0 0 0 1</u>	$A + RAM(B) \rightarrow A$	None	Add RAM to A
ADT		4A	<u>0 1 0 0</u>   <u>1 0 1 0</u>	$A + 10_{10} \rightarrow A$	None	Add Ten to A
AISC	y	5-	<u>0 1 0 1</u>   y	$A + y \rightarrow A$	Carry	Add Immediate, Skip on Carry (y $\neq$ 0)
CASC		10	<u>0 0 0 1</u>   <u>0 0 0 0</u>	$\bar{A} + RAM(B) + C \rightarrow A$ Carry $\rightarrow C$	Carry	Complement and Add with Carry, Skip on Carry
CLRA		00	<u>0 0 0 0</u>   <u>0 0 0 0</u>	$0 \rightarrow A$	None	Clear A
COMP		40	<u>0 1 0 0</u>   <u>0 0 0 0</u>	$\bar{A} \rightarrow A$	None	Ones complement of A to A
NOP		44	<u>0 1 0 0</u>   <u>0 1 0 0</u>	None	None	No Operation
RC		32	<u>0 0 1 1</u>   <u>0 0 1 0</u>	"0" $\rightarrow C$	None	Reset C
SC		22	<u>0 0 1 0</u>   <u>0 0 1 0</u>	"1" $\rightarrow C$	None	Set C
XOR		02	<u>0 0 0 0</u>   <u>0 0 1 0</u>	$A \oplus RAM(B) \rightarrow A$	None	Exclusive-OR RAM with A
<b>TRANSFER OF CONTROL INSTRUCTIONS</b>						
JID		FF	<u>1 1 1 1</u>   <u>1 1 1 1</u>	ROM (PC <sub>9:8</sub> ,A,M) $\rightarrow$ PC <sub>7:0</sub>	None	Jump Indirect (Note 3)
JMP	a	6-	<u>0 1 1 0</u>   <u>0 0 a<sub>9:8</sub></u> a <sub>7:0</sub>	a $\rightarrow$ PC	None	Jump
JP	a	--	<u>1</u>   a <sub>6:0</sub> (pages 2,3 only)	a $\rightarrow$ PC <sub>6:0</sub>	None	Jump within Page (Note 4)
			<u>1 1</u>   a <sub>5:0</sub> (all other pages)	a $\rightarrow$ PC <sub>5:0</sub>		
JSRP	a	--	<u>1 0</u>   a <sub>5:0</sub>	PC+1 $\rightarrow$ SA $\rightarrow$ SB $\rightarrow$ SC 0010 $\rightarrow$ PC <sub>9:6</sub> a $\rightarrow$ PC <sub>5:0</sub>	None	Jump to Subroutine Page (Note 5)
JSR	a	6-	<u>0 1 1 0</u>   <u>1 0 a<sub>9:8</sub></u> a <sub>7:0</sub>	PC+1 $\rightarrow$ SA $\rightarrow$ SB $\rightarrow$ SC a $\rightarrow$ PC	None	Jump to Subroutine
RET		48	<u>0 1 0 0</u>   <u>1 0 0 0</u>	SC $\rightarrow$ SB $\rightarrow$ SA $\rightarrow$ PC	None	Return from Subroutine
RETSK		49	<u>0 1 0 0</u>   <u>1 0 0 1</u>	SC $\rightarrow$ SB $\rightarrow$ SA $\rightarrow$ PC	Always Skip on Return	Return from Subroutine then Skip

Table 3.1 COP420 Series/COP444L Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>MEMORY REFERENCE INSTRUCTIONS</b>						
CAMQ		33 3C	$\begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$	A → Q7:4 RAM(B) → Q3:0	None	• Copy A, RAM to Q
CQMA		33 2C	$\begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$	Q7:4 → RAM(B) Q3:0 → A	None	• Copy Q to RAM, A
LD	r	-5	$\begin{bmatrix} 0 & 0 &   & r &   & 0 & 1 & 0 & 1 \end{bmatrix}$	RAM(B) → A Br ⊕ r → Br	None	Load RAM into A, Exclusive-OR Br with r
LDD	r,d	23 --	$\begin{bmatrix} 0 & 0 & 1 & 0 &   & 0 & 0 & 1 & 1 \\ 0 & 0 &   & r &   & d \end{bmatrix}$	RAM(r,d) → A	None	• Load A with RAM pointed to directly by r,d
LQID		BF	$\begin{bmatrix} 1 & 0 & 1 & 1 &   & 1 & 1 & 1 & 1 \end{bmatrix}$	ROM(PC9:8,A,M) → Q SB → SC	None	Load Q Indirect (Note 3)
RMB	0 1 2 3	4C 45 42 43	$\begin{bmatrix} 0 & 1 & 0 & 0 &   & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 &   & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 &   & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 &   & 0 & 0 & 1 & 1 \end{bmatrix}$	0 → RAM(B) <sub>0</sub> 0 → RAM(B) <sub>1</sub> 0 → RAM(B) <sub>2</sub> 0 → RAM(B) <sub>3</sub>	None	Reset RAM Bit
SMB	0 1 2 3	4D 47 46 4B	$\begin{bmatrix} 0 & 1 & 0 & 0 &   & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 &   & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 &   & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 &   & 1 & 0 & 1 & 1 \end{bmatrix}$	1 → RAM(B) <sub>0</sub> 1 → RAM(B) <sub>1</sub> 1 → RAM(B) <sub>2</sub> 1 → RAM(B) <sub>3</sub>	None	Set RAM Bit
STII	y	7-	$\begin{bmatrix} 0 & 1 & 1 & 1 &   & y \end{bmatrix}$	y → RAM(B) Bd + 1 → Bd	None	Store Memory Immediate and Increment Bd
X	r	-6	$\begin{bmatrix} 0 & 0 &   & r &   & 0 & 1 & 1 & 0 \end{bmatrix}$	RAM(B) ↔ A Br ⊕ r → Br	None	Exchange RAM with A, Exclusive-OR Br with r
XAD	r,d	23 --	$\begin{bmatrix} 0 & 0 & 1 & 0 &   & 0 & 0 & 1 & 1 \\ 1 & 0 &   & r &   & d \end{bmatrix}$	RAM(r,d) ↔ A	None	• Exchange A with RAM pointed to directly by r,d
XDS	r	-7	$\begin{bmatrix} 0 & 0 &   & r &   & 0 & 1 & 1 & 1 \end{bmatrix}$	RAM(B) ↔ A Bd - 1 → Bd Br ⊕ r → Br	Bd decrements past 0	Exchange RAM with A and Decrement Bd, Exclusive-OR Br with r
XIS	r	-4	$\begin{bmatrix} 0 & 0 &   & r &   & 0 & 1 & 0 & 0 \end{bmatrix}$	RAM(B) ↔ A Bd + 1 → Bd Br ⊕ r → Br	Bd increments past 15	Exchange RAM with A and Increment Bd, Exclusive-OR Br with r
<b>REGISTER REFERENCE INSTRUCTIONS</b>						
CAB		50	$\begin{bmatrix} 0 & 1 & 0 & 1 &   & 0 & 0 & 0 & 0 \end{bmatrix}$	A → Bd	None	Copy A to Bd
CBA		4E	$\begin{bmatrix} 0 & 1 & 0 & 0 &   & 1 & 1 & 1 & 0 \end{bmatrix}$	Bd → A	None	Copy Bd to A
LBI	r,d	--	$\begin{bmatrix} 0 & 0 &   & r &   & (d-1) \\ (d = 0, 9:15) \end{bmatrix}$ or $\begin{bmatrix} 0 & 0 & 1 & 1 &   & 0 & 0 & 1 & 1 \\ 1 & 0 &   & r &   & d \end{bmatrix}$ (any d)	r,d → B	Skip until not a LBI	Load B Immediate with r,d (Note 6)
LEI	y	33 6-	$\begin{bmatrix} 0 & 0 & 1 & 1 &   & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 &   & y \end{bmatrix}$	y → EN	None	• Load EN Immediate (Note 7)
XABR		12	$\begin{bmatrix} 0 & 0 & 0 & 1 &   & 0 & 0 & 1 & 0 \end{bmatrix}$	A ↔ Br (0,0 → A <sub>3</sub> ,A <sub>2</sub> )	None	Exchange A with Br

Table 3.1 COP420 Series/COP444L Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>TEST INSTRUCTIONS</b>						
SKC		20	0010 0000		C = "1"	Skip if C is True
SKE		21	0010 0001		A = RAM(B)	Skip if A Equals RAM
SKGZ		33	0011 0011		G <sub>3:0</sub> = 0	* Skip if G is Zero (all 4 bits)
		21	0010 0001			
SKGBZ		33	0011 0011	1st byte		* Skip if G Bit is Zero
	0	01	0000 0001	} 2nd byte	G <sub>0</sub> = 0	
	1	11	0001 0001		G <sub>1</sub> = 0	
	2	03	0000 0011		G <sub>2</sub> = 0	
	3	13	0001 0011		G <sub>3</sub> = 0	
SKMBZ	0	01	0000 0001		RAM(B) <sub>0</sub> = 0	Skip if RAM Bit is Zero
	1	11	0001 0001		RAM(B) <sub>1</sub> = 0	
	2	03	0000 0011		RAM(B) <sub>2</sub> = 0	
	3	13	0001 0011		RAM(B) <sub>3</sub> = 0	
SKT		41	0100 0001		A time-base counter carry has occurred since last test	Skip on Timer (Note 3)
<b>INPUT/OUTPUT INSTRUCTIONS</b>						
ING		33	0011 0011	G → A	None	* Input G Ports to A
		2A	0010 1010			
ININ		33	0011 0011	IN → A	None	* Input IN Inputs to A (Note 2)
		28	0010 1000			
INIL		33	0011 0011	IL <sub>3</sub> , "1", "0", IL <sub>0</sub> → A	None	* Input IL Latches to A (Note 3)
		29	0010 1001			
INL		33	0011 0011	L <sub>7:4</sub> → RAM(B) L <sub>3:0</sub> → A	None	* Input L Ports to RAM, A
		2E	0010 1110			
OBD		33	0011 0011	Bd → D	None	* Output Bd to D Outputs
		3E	0011 1110			
OGI	y	33	0011 0011	y → G	None	* Output to G Ports Immediate
		5-	0101  y			
OMG		33	0011 0011	RAM(B) → G	None	* Output RAM to G Ports
		3A	0011 1010			
XAS		4F	0100 1111	A ↔ SIO, C → SK	None	Exchange A with SIO (Note 3)

**Note 1:** All subscripts for alphabetical symbols indicate bit numbers unless explicitly defined (e.g., Br and Bd are explicitly defined). Bits are numbered 0 to N where 0 signifies the least significant (low-order, right-most bit). For example, A<sub>3</sub> indicates the most significant (left-most) bit of the 4-bit A register.

**Note 2:** The ININ instruction is not available on the 24-pin COP421 since this device does not contain the IN inputs.

**Note 3:** For additional information on the operation of the XAS, JID, LQID, INIL, and SKT instructions, see Section 3.2.

**Note 4:** The JP instruction allows a jump, while in subroutine pages 2 or 3, to any ROM location within the two-page boundary of pages 2 or 3. The JP instruction, otherwise, permits a jump to a ROM location within the current 64-word page. JP may not jump to the last word of a page.

**Note 5:** A JSRP transfers program control to subroutine page 2 (0010 is loaded into the upper 4 bits of P). A JSRP may not be used when in pages 2 or 3. JSRP may not jump to the last word in page 2.

**Note 6:** LBI is a single-byte instruction if d = 0, 9, 10, 11, 12, 13, 14, or 15. The machine code for the lower 4 bits equals the binary value of the "d" data minus 7, e.g., to load the lower four bits of B (Bd) with the value 9 (1001<sub>2</sub>), the lower 4 bits of the LBI instruction equal 8 (1000<sub>2</sub>). To load 0, the lower 4 bits of the LBI instruction should equal 15 (1111<sub>2</sub>).

**Note 7:** Machine code for operand field y for LEI instruction should equal the binary value to be latched into EN, where a "1" or "0" in each bit of EN corresponds with the selection or deselection of a particular function associated with each bit. (See Functional Description, EN Register.)



Table 3.2 provides a list of internal architecture, instruction operand and operational symbols used in the COP420-series/COP444L Instruction Set Table. Table 3.5 shows an alphabetical mnemonic index of COP420-series/COP444L instructions, indicating the hexadecimal opcode and description associated with each instruction. Table 3.6 is a list of COP420-series/COP444L instructions arranged in order of their hexadecimal opcodes.

The following text gives a description of each COP420-series/COP444L instruction, explaining the machine operations performed by each instruction and, where appropriate, providing short examples illustrating typical usage of particular instructions.

**Table 3.2. COP420-Series/COP444L Instruction Set Table Symbols**

Symbol	Definition
<b>INTERNAL ARCHITECTURE SYMBOLS</b>	
A	4-bit Accumulator
B	6-bit RAM Address Register
Br	Upper 2 bits of B (register address)
Bd	Lower 4 bits of B (digit address)
C	1-bit Carry Register
D	4-bit Data Output Port
EN	4-bit Enable Register
G	4-bit Register to latch data for G I/O Port
IL	Two 1-bit Latches associated with the IN <sub>3</sub> or IN <sub>0</sub> Inputs
IN	4-bit Input Port
L	8-bit TRI-STATE I/O Port
M	4-bit contents of RAM Memory pointed to by B Register
PC	10-bit ROM Address Register (program counter)
Q	8-bit Register to latch data for L I/O Port
SA	10-bit Subroutine Save Register A
SB	10-bit Subroutine Save Register B
SC	10-bit Subroutine Save Register C
SIO	4-bit Shift Register and Counter
SK	Logic-Controlled Clock Output
<b>Symbol</b>	<b>Definition</b>
<b>INSTRUCTION OPERAND SYMBOLS</b>	
d	4-bit Operand Field, 0–15 binary (RAM Digit Select)
r	2-bit Operand Field, 0–3 binary (RAM Register Select)
a	10-bit Operand Field, 0–1023 binary (ROM Address)
y	4-bit Operand Field, 0–15 binary (Immediate Data)
RAM(s)	Contents of RAM location addressed by s
ROM(t)	Contents of ROM location addressed by t
<b>OPERATIONAL SYMBOLS</b>	
+	Plus
–	Minus
→	Replaces
↔	Is exchanged with
=	Is equal to
A	The ones complement of A
⊕	Exclusive-OR
:	Range of values

## 3.2 COP420-Series/COP444L Instruction Set Description

### Arithmetic Instructions

**ASC** (Add with carry, Skip on Carry) performs a binary addition of A, C (Carry bit), and M, placing the result in A and C. If a carry occurs, the next program instruction is skipped.

**ADD** (ADD) performs binary addition. The 4-bit addends are A and M. The 4-bit sum is placed in A. ADD does not affect the carry or skip.

**ADT** (ADD Ten to A) adds ten (1010<sub>2</sub>) to A and, like ADD, does not affect the carry or skip. It is intended to facilitate Binary Coded Decimal (BCD) arithmetic. For example, the following sequence of instructions will perform a single-digit BCD add of the contents of A and M [the carry is assumed set when entering this routine if addition of the previous least significant digits produced an overflow (A > 9)]:

```
AISC 6
ASC
ADT
```

The AISC 6 instruction adds a BCD correction factor (i.e., 6) to the digit in the accumulator. (See AISC instruction.) Since the accumulator contains a BCD digit ( $\leq 9$ ) no carry will occur and the next instruction, ASC, will always be executed. The ASC instruction adds the carry and memory digit to A, as explained above. If the result does *not* produce a carry, signifying that the previous AISC 6 (correction factor) instruction was unnecessary, the ADT instruction is executed, readjusting the accumulator to the proper BCD result. (Remember: ADT neither affects the carry nor skips.)

If the ASC result does produce a carry, C is set for propagation to the addition of the next most significant digits and, since no readjustment of the result is necessary, the ADT instruction is skipped.

**AISC** (Add Immediate, Skip on Carry) adds the instruction operand constant "y" (1–15) to A, skipping the next instruction if a carry out occurs (C is *not* changed). This instruction finds frequent use in BCD add and subtract routines (see ADT and CASC descriptions) as well as in testing the value of A. (If A is greater than 12, for instance, an AISC 5 will skip the next instruction.)

**CASC** (Complement and Add, Skip on Carry) performs a binary subtraction of A from M by summing the complement of A ( $\bar{A}$ ) with C and M, placing the result in A and C. If no carry out occurs, indicating a borrow, C is reset and the next instruction is executed. If a carry occurs, indicating no borrow, C is set and the next instruction is skipped.

A single BCD digit binary subtraction of A from M may be performed as follows. (The carry bit is assumed set upon initial entry to the routine.)

CASC  
ADT

The CASC instruction will set C and skip the ADT instruction if the subtraction does not result in a borrow ( $A > M$ ). If a borrow occurs, the ADT instruction is executed, readjusting the result to the proper BCD value, leaving C reset for propagation of the borrow in the subtraction of the next most significant BCD digits. CASC is functionally equivalent to a COMP instruction followed by an ASC.

CLRA (CLear A) clears the accumulator by placing zeros in each of the 4 bits of A.

This instruction is often required prior to loading A equal to a desired value with an AISC instruction if the previous contents of A are unknown. For instance, to load  $A = 11$ , the following sequence may be used:

CLRA  
AISC 11

The skip features associated with AISC need not be considered in this example. (A carry will never occur.)

COMP (COMPLement A) changes the state of each of 4 bits of A with ones becoming zeros and zeros becoming ones. It has the effect of, and may be used to perform, a binary (one's complement) subtraction of A from 15 ( $1111_2$ ), e.g., complementing  $A = 6$  ( $0110_2$ ) will yield 9 ( $1001_2$ ).

NOP (No OPERATION) does not perform any operation. It is useful, however, for simple single instruction time delays or to defeat the skip conditions associated with particular instructions.

SC (Set Carry) and RC (Reset Carry) set C and reset C, respectively. SC and RC are most often employed to initialize C prior to entering arithmetic routines. They also allow C to be used as a general-purpose (testable) flag, as long as subsequent instructions do not inadvertently affect the C register.

XOR (eXclusive-OR A with M) performs a logical EXCLUSIVE-OR operation of each bit of A with each corresponding bit of M, placing the result in A. This operation can be used to change the state of any bit in M, if the corresponding (equally weighted) bit of A is set. This follows from the EXCLUSIVE-OR truth table where a  $X + "1" = \bar{X}$ , and a  $X + "0" = X$ , assuming the "X" bits to be one of the 4 bits in M, and the "1" and "0" to be equally weighted bits in A. This instruction, therefore, allows the selective complementing or toggling of one or more bits of M. Example: to change the state of bit 2 of M, set  $A = 0100$ , perform an XOR, then exchange A into M with an X instruction.

## Input/Output Instructions

ING (INput G ports to A) transfers the 4-bit contents of the IN ports ( $IN_3 - IN_0$ ) to A.

ININ (INput IN inputs to A) transfers the 4-bit contents of the IN ports ( $IN_3 - IN_0$ ) to A.

INIL (INput IL latches to A) is a special purpose instruction which inputs the two latches  $IL_3$  and  $IL_0$  (see Figure 3.1 below) and, if the appropriate option is selected, a general-purpose input, CKO, to the accumulator — the unused bit/bits of A are reset. Specifically, INIL places  $IL_3 \rightarrow A_3$ ,  $CKO \rightarrow A_2$ , "0"  $\rightarrow A_1$ ,  $IL_0 \rightarrow A_0$ .  $IL_3$  and  $IL_0$  are the outputs of latches associated with the  $IN_3$  and  $IN_0$  inputs. (The *general purpose* inputs,  $IN_3 - IN_0$ , are input to A upon the execution of an ININ instruction. (See ININ Instruction.) The  $IL_3$  and  $IL_0$  latches are set if a low-going pulse ("1" to "0") has occurred on the  $IN_3$  and  $IN_0$  inputs, respectively, since the last INIL instruction, provided the input pulse stays low for at least two instruction times. Execution of an INIL inputs  $IL_3$  and  $IL_0$  into  $A_3$  and  $A_0$  respectively, and resets these latches to allow them to respond to subsequent low-going pulses on the  $IN_3$  and  $IN_0$  lines. These latches are not cleared during a power on reset.

If CKO is mask-programmed as a general-purpose input, an INIL will input the state of CKO into  $A_2$ . If CKO has not been so programmed, a "1" will be placed in  $A_2$ . A "0" is always placed in  $A_1$  upon the execution of an INIL.

INIL is useful in recognizing and capturing pulses of short duration or which can't be read conveniently by an ININ instruction.

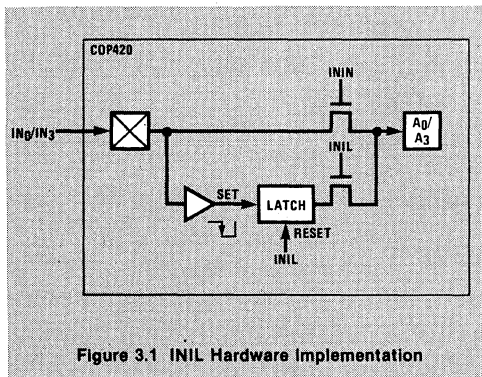


Figure 3.1 INIL Hardware Implementation

INL (INput L ports to M, A) transfers the 8-bit contents of the bidirectional TRI-STATE® I/O ports to M.  $L_7 - L_4$  are placed in  $M_3 - M_0$  (the memory digit pointed to by the B register);  $L_3 - L_0$  are placed in  $A_3 - A_0$ .

**OBD** (Output Bd to D outputs) transfers the 4-bit contents of Bd (lower 4 bits of the B register) to the D output ports (D<sub>3</sub>-D<sub>0</sub>). Since, in many applications, the D outputs are connected to a digit decoder, the direct output of Bd allows for a standard interconnect to the binary inputs of the decoder/driver device.

**OGI** (Output to G ports Immediate) transfers the four bits specified in the "y" operand field of this instruction (0-15, binary) to G<sub>3</sub>-G<sub>0</sub>.

**OMG** (Output M to G ports) transfers the 4-bit contents of M (M<sub>3</sub>-M<sub>0</sub>) to G<sub>3</sub>-G<sub>0</sub>.

**XAS** (eXchange A with SIO) exchanges the 4-bit contents of A (A<sub>3</sub>-A<sub>0</sub>) with the 4-bit contents of the SIO register (SIO<sub>3</sub>-SIO<sub>0</sub>). SIO will contain serial-in/serial-out shift register or binary counter data, depending on the value of the EN register. An XAS instruction will also affect the SK output. The XAS instruction copies C into the SKL latch. In the counter mode, SK is the output of SKL; in the shift register mode, SK outputs SKL ANDed with the clock.

For further information on the EN register and its relationship to the XAS instruction, see LEI Instruction, below. If SIO is selected as a shift register, an XAS instruction must be performed once every 4 instruction cycle times to effect a continuous serial-in or serial-out data stream.

### Transfer of Control Instructions

**JID** (Jump InDirect) is an indirect addressing instruction, transferring program control to a new ROM location addressed by the *contents* of the ROM location pointed to by A and M. Specifically, it loads the lower 8 bits of the ROM address register P with the *contents* of ROM pointed to by the 10-bit word P<sub>9</sub>P<sub>8</sub>A<sub>3</sub>A<sub>2</sub>A<sub>1</sub>A<sub>0</sub>M<sub>3</sub>M<sub>2</sub>M<sub>1</sub>M<sub>0</sub>. The contents of the selected ROM location (I<sub>7</sub>-I<sub>0</sub>) are, therefore, loaded into P<sub>7</sub>-P<sub>0</sub>, changing the lower 8 bits of P to transfer program control to the new ROM location.

P<sub>9</sub> and P<sub>8</sub> remain unchanged throughout the execution of the JID instruction. JID, therefore, may only jump to a ROM location within the current 4-page ROM "block" (pages 0-3, 4-7, 8-11 or 12-15). For further information regarding the "paging" restrictions associated with the JID instruction, see Section 4.1.

JID can be useful in keyboard-decode routines when the values associated with the row and column of a particular key closure are placed in A and M for a jump indirect to the contents of ROM which point to the starting address of the appropriate routine associated with that particular key closure. For an example of use of the JID instruction to access a keyboard-decode ROM pointer table, see Display/Keyboard Program, Section 5.3, #16.

**JMP** (JuMP) transfers program control to any word in the ROM as specified by the "a" field of this instruction. The 10-bit "a" field is placed in P<sub>9</sub>-P<sub>0</sub>. JMP is used to transfer program control from one page to *another* page (if in page 2 or 3, the more efficient single-byte JP instruction may be used) or to transfer control to the *last* word of the current page — an invalid transfer for the JP instruction.

**JP** (Jump within Page) transfers program control to the ROM address specified in the operand field of this instruction. The machine code and operand field of this instruction have two formats. If program execution is currently within page 2 or 3 (subroutine pages) a 7-bit "a" field is specified, transferring program control to a word within either of the two subroutine pages. Otherwise, only a 6-bit "a" field is specified, transferring program control to a particular word within the *current* 64-word ROM page.

Specifically, this instruction places a<sub>6</sub>-a<sub>0</sub> in P<sub>6</sub>-P<sub>0</sub> if the program is currently in subroutine page 2 or 3. If in any other page, it places a<sub>5</sub>-a<sub>0</sub> in P<sub>5</sub>-P<sub>0</sub>.

The restrictions associated with the JP instruction, therefore, are that a 7-bit "a" field may be used only when in pages 2 or 3. Otherwise, a JP may be used only to jump within the current page by specifying a 6-bit "a" field in the operand of this instruction. An additional restriction associated with the JP instruction, in either of the above two formats, is that a JP to the last word of any page is invalid, i.e., "a" may not equal all 1s. A transfer of program control to last word on a page may be effected by using a JMP instruction. (See JMP Instruction, above.)

**JSRP** (Jump to SubRoutine Page) is used to transfer program control from a page other than 2 or 3 to a word within page 2. It accomplishes this by placing a 2 (0010<sub>2</sub>) in P<sub>9</sub>-P<sub>6</sub>, and the word address specified in the 6-bit "a" field of the instruction into P<sub>5</sub>-P<sub>0</sub>. Designed to transfer control to subroutines, it *pushes the stack* to save the subroutine return address — the address of the next program instruction is saved in SA and the other subroutine-save registers are likewise pushed (P + 1 → SA → SB → SC). Any previous contents of SC are lost, since SC is the last of the three subroutine-save registers. Subroutine nesting, therefore, is permitted to three levels. JSRP is used in conjunction with the RET or RETSK instructions which "pop" the stack at the end of subroutine to return program control to the main program. As with the JP instruction, JSRP may not transfer program control to the last word of page 2: "a" may not equal all "1s." A JSR may be used to jump to the last word of a subroutine beginning at the last word of page 2. (See JSR, below.) As mentioned above, a further restriction is that a

JSRP may not be used when in subroutine pages 2 or 3. To transfer program control to a subroutine in page 2 when in pages 2 or 3, the double-byte JSR should be used, or, if it is not necessary to push the stack, a JP instruction may be used.

**JSR** (Jump to SubRoutine) transfers program control to a subroutine located at a particular word address in *any* ROM page. It modifies the entire P register with the value of the "a" operand of this instruction, as follows:  $a_9-a_0 \rightarrow P_9-P_0$ . As with the JSRP instruction, JSR pushes the stack ( $P+1 \rightarrow SA \rightarrow SB \rightarrow SC$ ), saving the next program instruction for a return from the subroutine to the main program via a RET or RETSK instruction. JSR may be used to overcome the restrictions associated with the JSRP instruction: to jump to a subroutine and push the stack when in pages 2 or 3, or to jump to a subroutine located at the last word of page 2.

**RET** (RETurn from subroutine) is used to return program control to the main program following a JSR or JSRP instruction. RET "pops" the stack ( $SC \rightarrow SB \rightarrow SA \rightarrow P$ ): the next main program instruction address ( $P+1$ ) saved in SA is loaded into P, the contents of SB are loaded into SA and the contents of SC are loaded into SB. (The contents of SC are also retained in SC.) Program control, therefore, is returned to the instruction immediately following the previous subroutine call.

**RETSK** (RETurn from subroutine then SKip), as with the RET instruction above, pops the stack ( $SC \rightarrow SB \rightarrow SA \rightarrow P$ ), restoring program control to the main program following a subroutine call. It, however, *always* skips the first instruction encountered when it returns to the main program. This instruction, therefore, provides the programmer with an alternate return from subroutines, either via a RET or RETSK, based upon tests made within the subroutine itself.

**CAMQ** (Copy A, M to Q) transfers the 8-bit contents of A and M to the Q latches.  $A_3-A_0$  are output to  $Q_7-Q_4$ ;  $M_3-M_0$  are output to  $Q_3-Q_0$ . Note that CAMQ is the inverse of CQMA (see CQMA Instruction, below) with respect to the 4 bits of Q with which A and M communicate. Therefore, the input and processing of Q must often be followed by an X (Exchange M with A) instruction before final output to Q in order to maintain the proper bit-weights of the Q data. For example, the following instructions read Q to M, A, set  $Q_7$  and perform the necessary exchange before execution of the CAMQ instruction:

```
CQMA      ; Q TO M, A
SMB 3     ; SET  $Q_7$  BIT LOCATED IN  $M_3$ 
X         ; EXCHANGE M WITH A
CAMQ     ; A, M TO Q
```

**CQMA** (Copy Q to M, A) transfers the 8-bit contents of the Q latches to M and A.  $Q_7-Q_4$  are placed in  $M_3-M_0$ ;  $Q_3-Q_0$  are placed in  $A_3-A_0$ . CQMA can be employed after an LQID (Load Q InDirect) instruction to input or alter the value of lookup data. CQMA is also an essential instruction when the COP420 is employed as a MICROBUS™ peripheral component. In such applications,  $IN_3$  is used by the control microprocessor to write bus data from the L ports to the Q latches. (See Section 2.4, MICROBUS™ option.) A CQMA will then input this data to M, A as explained above for processing by the COP420 program.

### Memory Reference Instructions

**LD** (LoaD M into A) loads M (the 4-bit contents of RAM pointed to by the B register:  $M_3-M_0$ ) into  $A_3-A_0$ . After M is loaded into A, the 2-bit "r" operand field is EXCLUSIVE-ORed with the contents of Br (upper 2 bits of B — RAM register select) to point to a new RAM register for successive memory reference operations. Since the properties of the EXCLUSIVE-OR logic operation are such that a  $1 \oplus X$  equals the complement of X, use of the "r" field allows the programmer to switch between any one of the 4 RAM registers by complementing the appropriate bit/bits of the current contents of the Br register. Of course, if "r" = 0, the contents of Br will remain unchanged after the execution of a LD instruction.

For example, if the assembly language instruction LD 3 ("r" =  $11_2$ ) is executed with  $Br = 2$  ( $10_2$ ) and  $Bd = 12$  ( $1100_2$ ), the contents of RAM register 2, digit 12 will be loaded to A and Br will be changed to ( $11_2 + 10_2 = 01_2$ ), with B pointing to RAM register 1, digit 12. For assembly language programming use of an EXCLUSIVE-OR "r" operand field with memory reference instructions which use this field is optional — if not specified, an "0" operand is assumed. For further information on allocating RAM map locations for optimum use of the EXCLUSIVE-OR feature associated with this and other memory reference instructions and for sample routines utilizing this feature, refer to Sections 4.2 and 4.4.

**SMB** (Set Memory Bit) and **RMB** (Reset Memory Bit) set and reset, respectively, a bit in M as specified by the operand field of these instructions. (Remember: M is the 4-bit RAM digit pointed to by the B register.) The operand field is specified according to the bit number (0-3, left-most to right-most bit) of the particular bit to be set or reset, e.g., an SMB 3 would set the most significant bit of M. These instructions are useful in operating upon program status flags located in RAM.

**STII** (Store Memory Immediate and Increment Bd) loads the 4-bit contents specified by the "y"

operand field of the instruction into the RAM memory digit pointed to by the B register,  $M_3-M_0$ . It is important to note that the value of Bd (RAM digit-select) is *incremented* (as with the XIS instruction) after the "y" data is stored in M.

**LDD** (Load A with M Directly) loads the 4-bit contents of the RAM memory location pointed to directly by the "r" and "d" operand fields (register and digit select, respectively) of the instruction,  $M_3-M_0$ , into  $A_3-A_0$ . Note that this instruction and the XAD instruction differ from other memory reference instructions in that the operand of the instruction, not the B register, is used to point to the appropriate RAM digit location to be accessed — the B register is unaffected by these instructions. This instruction is useful in accessing RAM counters, status and flag digits, etc., within routines or loops without destroying the previous value of B, allowing the latter to be used for sequential memory access operations and for other reiterative purposes.

**LQID** (Load Q INdirect) is, in effect, a ROM data "lookup" instruction. It transfers the 8-bit contents of ROM,  $I_7-I_0$ , pointed to by the 10-bit word  $P_9P_8AM$  to  $Q_7-Q_0$ , respectively. It does this by pushing the stack ( $P+1 \rightarrow SA \rightarrow SB \rightarrow SC$ ) and replacing the least significant 8 bits of P as follows:  $A_3-A_0 \rightarrow P_7-P_4$ ;  $M_3-M_0 \rightarrow P_3-P_0$ , leaving the two most significant bits of P unchanged. The ROM data pointed to by the new P address is fetched and loaded into the Q latches,  $Q_7-Q_0$ . Next, the stack is popped ( $SC \rightarrow SB \rightarrow SA \rightarrow P$ ), restoring the previous pushed value of P ( $P+1$ ) to continue sequential program execution. Since LQID pushes  $SB \rightarrow SC$ , the previous contents of SC are lost. Also, when LQID pops the stack, the previously pushed contents of SB are left in SC as well as loaded back into SB. The net result, therefore, of an LQID instruction upon the subroutine-save stack is that the contents of SB are placed in SC ( $SB \rightarrow SC$ ). Since it pushes the stack, a LQID should not be executed when *three* levels of subroutine nesting are currently in effect. (The last return address in SC will be lost.)

Since, as with the JID instruction, LQID affects only the lower 8 bits of P ( $P_9$  and  $P_8$  are unchanged), it may only access ROM data located within the current 4-page ROM "block" (pages 0-3, 4-7, 8-11 or 12-15). For further information on the use of the LQID instruction, see Section 4.1.

**X** (eXchange M with A) exchanges the 4-bit contents of RAM pointed to by the B register,  $M_3-M_0$ , with  $A_3-A_0$ . The "r" operand field of the instruction is EXCLUSIVE-ORed with the contents of Br after the exchange to provide a new Br RAM register select value as explained in the LD instruction above.

**XAD** (eXchange A with M Directly) exchanges the 4-bit contents of the RAM memory location pointed

to directly by the "r" and "d" operand fields of the instruction,  $M_3-M_0$ , with  $A_3-A_0$ . It has the same characteristics and utility as the LDD instruction above, e.g., the B register is not affected.

**XDS** (eXchange M with A, Decrement Bd and Skip on borrow) performs the same operation as the X instruction above, and also decrements the value of the Bd register (RAM digit-select) *after* the exchange. Use of an "r" operand field will, therefore, result in both an altered RAM *digit*-select value and a new RAM *register* select value in B. XDS skips the next program instruction when Bd is decremented *past* 0 (after the contents of RAM digit 0 have been exchanged with A and XDS decrements Bd to 15). Repeated XDSs will "walk down" through the digits of a RAM register before skipping. XDS together with X instructions can be used to operate upon the corresponding digits of different RAM registers in successive fashion. (See Section 4.2.)

**XIS** (eXchange M with A, Increment Bd, and Skip on carry) performs the same operation as the XDS instruction except that it *increments* Bd *after* the exchange and skips the next program instruction after Bd increments *past* 15 (after the contents of RAM digit 15 have been exchanged with A and XIS increments Bd to 0). Consequently, successive XISs "walk up" through the digits of a RAM register before skipping.

#### Register Reference Instructions

**CAB** (Copy A to Bd) transfers the 4-bit contents of A,  $A_3-A_0$ , to Bd (the RAM digit-select register). This instruction allows the loading of a new RAM digit-select value via the accumulator, a useful operation in many memory-digit access loops.

**CBA** (Copy Bd to A) transfers the 4-bit contents of Bd (RAM digit select) to  $A_3-A_0$ . It is the functional complement of the CAB instruction and finds similar use in memory-digit access loops.

**LBI** (Load B Immediate) loads the B register with the 6-bit value specified by the "r" (2-bit) and "d" (4-bit) fields of the instruction. Its purpose is to directly load a new RAM register and digit select value into B and, unlike CAB, CBA or XABR, does not require use of the accumulator. A further distinction with respect to CAB and CBA is its ability to alter the Br register (RAM register-select).

The LBI instruction is coded or assembled into machine language as *either* a single- or a double-byte instruction, depending on the value of the "d" field. If the "d" field value equals 0 or 9 through 15, the instruction is coded as a single-byte instruction with the lower 6 bits equal to the value of "d" *minus* 1. If the "d" field equals 1 through 8 (1-8), the instruction is coded as a double-byte instruction, with the lower 6 bits of the second byte equal to the value of "d." (See LBI Instruction, Table 3.1, and Note 6 of Table 3.1.)



To take advantage of the more efficient single-byte LBI format, frequently used program data (counters, flags, etc.) should be placed within RAM digit locations accessible by the LBI single-byte "d" field (d = 0, 9-15). (See Section 4.2 for further information.)

An important characteristic of the LBI instruction is that it will skip all subsequent LBI instructions until it encounters an instruction which is not an LBI. This feature accommodates it for use in multiple-entry subroutines. (For example, see Adjacent Memory Move Routine, Section 4.4.)

LEI (Load EN Immediate) loads the enable register with the value contained in the "y" operand field of this instruction (0-15, binary). Its function is to select or deselect a particular software selectable feature associated with each of the four bits of the enable register (EN<sub>3</sub>-EN<sub>0</sub>). These features and the corresponding bit-weights and values associated with each feature are as follows:

1. The least significant bit of the enable register, EN<sub>0</sub>, selects the SIO register as either a 4-bit shift register or a 4-bit binary counter.

With EN<sub>0</sub> set, SIO is an asynchronous binary counter, decrementing its value by one upon each low-going pulse ("1" to "0") occurring on the SI input. Each pulse must remain at each logic level at least two instruction cycles. SK outputs the value of the C upon the execution of an XAS and remains latched until the execution of another XAS instruction. The SO output is equal to the value of EN<sub>3</sub>.

With EN<sub>0</sub> reset, SIO is a serial shift register, shifting continuously left each instruction cycle time. The data present at SI goes into the least significant bit of SIO; SO can be enabled to output the most significant bit of SIO each cycle time. SK output becomes a logic-controlled clock, providing a SYNC signal each instruction time. It will start outputting a SYNC pulse upon the execution of an XAS instruction with C = "1," stopping upon the execution of a subsequent XAS with C = "0."

If EN<sub>0</sub> is changed from "1" to "0" ("0" to "1"), the SK output will change from "1" to SYNC (SYNC to "1") *without* the execution of an XAS instruction.

2. With EN<sub>1</sub> set, the IN<sub>1</sub> input is enabled as an interrupt input. Upon the occurrence of a negative pulse on IN<sub>1</sub>, program control is transferred to the last word of page 3 (address OFF<sub>16</sub>). Immediately following an interrupt, EN<sub>1</sub> is reset to disable further interrupts until later set by an LEI instruction (usually at the end of the interrupt service routine or later within the main program).

The following features are associated with the IN<sub>1</sub> interrupt procedure and protocol and must be considered by the programmer when utilizing this software-selectable feature of the COP420-series. (Interrupt is unavailable on the COP421-series since it does not have the IN<sub>3</sub>-IN<sub>0</sub> inputs.)

- a. The interrupt, once acknowledged as explained below, pushes the next sequential program counter address (P + 1) onto the stack, pushing in turn the contents of the other subroutine-save registers to the next lower level (P + 1 → SA → SB → SC). Any previous contents of SC are lost. The program counter is set to address OFF<sub>16</sub> (the last word of page 3) and EN<sub>1</sub> is reset.
- b. An interrupt will be acknowledged only after the following conditions are met:
  - 1) EN<sub>1</sub> has been set;
  - 2) A low-going pulse ("1" to "0") at least two instruction cycles in width has occurred on the IN<sub>1</sub> input;
  - 3) A currently executing instruction has been completed;
  - 4) All successive transfer of control instructions and successive LBIs have been completed (e.g., if the main program is executing a JP instruction which transfers program control to another JP instruction, the interrupt will not be acknowledged until the second JP instruction has been executed).
- c. Upon acknowledgement of an interrupt, the skip logic status is saved and implemented upon the execution of a subsequent RET instruction. For example, if an interrupt occurs during the execution of ASC (Add with carry, Skip on Carry) instruction which results in a carry, the next instruction (which would normally be skipped) is *not* skipped; instead, its address is pushed onto the stack, the skip logic status is saved and program control is transferred to the interrupt servicing routine at location OFF<sub>16</sub>. At the *end* of the interrupt routine, a RET instruction is executed to pop the stack and return program control to the instruction following the original ACS. *At this time*, the skip logic is enabled and skips this instruction because of the previous ASC carry. Since, as explained above, it is the RET instruction which enables the previously saved status of the skip logic, subroutines should not be nested within the interrupt service routine since their RET instruction will enable any previously saved main program skips, interfering with the orderly execution of the interrupt routine.
- d. The first instruction of the interrupt routine at address OFF<sub>16</sub> must be NOP.

3. With EN<sub>2</sub> set, the L drivers are enabled, loading data previously latched into Q to the L I/O ports. Resetting EN<sub>2</sub> disables the L drivers, placing the L I/O ports in a high-impedance state. When the L I/O ports are used as segment drivers to an LED display, the setting and resetting of EN<sub>2</sub> results in the outputting and blanking, respectively, of segment data to the display. When using the MICROBUS™ option EN<sub>2</sub> does not affect the L drivers.
4. EN<sub>3</sub>, in conjunction with EN<sub>0</sub>, affects the SO output. With EN<sub>0</sub> set (binary counter option selected) SO will output the value loaded into EN<sub>3</sub>. With EN<sub>0</sub> reset (serial shift register feature selected), setting EN<sub>3</sub> enables SO as the output of the SIO shift register, outputting serial shifted data (the most significant bit of SIO) each instruction time as explained above. Resetting EN<sub>3</sub> with the serial shift register feature selected disables SO as the shift register output: data continues to be shifted through SIO and can be exchanged with A via an XAS instruction but SO remains reset to "0." Figure 3.2 below provides a summary of the features associated with EN<sub>3</sub> and EN<sub>0</sub>.

EN <sub>3</sub>	EN <sub>0</sub>	SIO	SI	SO	SK after XAS
0	0	Shift Register	Input to Shift Register	0	If SKL = 1, SK = SYNC If SKL = 0, SK = 0
1	0	Shift Register	Input to Shift Register	Serial Out	If SKL = 1, SK = SYNC If SKL = 0, SK = 0
0	1	Binary Counter	Negative Edge Sensitive Input to Binary Counter	0	If SKL = 1, SK = 1 If SKL = 0, SK = 0
1	1	Binary Counter	Negative Edge Sensitive Input to Binary Counter	1	If SKL = 1, SK = 1 If SKL = 0, SK = 0

Figure 3.2 Enable Register Features — Bits EN<sub>3</sub> and EN<sub>0</sub>

**XABR** (eXchange A with Br) exchanges Br (upper 2 bits of B: RAM register-select) with A. Since Br contains only 2 bits, only the lower two bits of A, A<sub>1</sub>-A<sub>0</sub>, are placed in Br. Similarly, the 2 bits of Br are placed in A<sub>1</sub>-A<sub>0</sub> with "0s" being loaded into the upper 2 bits of A, A<sub>3</sub>-A<sub>2</sub>. XABR is an efficient means of loading the Br register via the accumulator — a direct load of the Br register must otherwise be accomplished by an LBI instruction which also affects the Bd portion of the B register.

### Test Instructions

**SKC** (SKip on Carry) skips the next program instruction if the carry bit is equal to "1." When used in conjunction with the RC and SC instructions, it allows C to be used as a 1-bit testable flag.

**SKE** (SKip if A Equals M) compares all 4 bits of A with M, skipping the next instruction if the value of A is equal to the value of M. SKE can be used to compare A with a status or counter digit in M, skipping to an instruction which transfers program control to another routine if equality exists.

**SKGBZ** (SKip if G Bit is Zero) is a double-byte instruction. It tests the state of *one* of the four G lines (G<sub>3</sub>-G<sub>0</sub>) as specified by the "n" operand of the instruction, skipping the next program instruction if the specified G line is equal to "0."

**SKGZ** (SKip if G is Zero) is a double-byte instruction. It tests the state of all *four* of the G lines, skipping the next program instruction if G<sub>3</sub>-G<sub>0</sub> are all equal to "0."

**SKMBZ** (SKip on Memory Bit Zero) skips the next program instruction if the RAM memory bit specified by the "n" field of the instruction (0-3, right-most to left-most M bit) is equal to "0." This instruction, together with the SMB and RMB instructions, allow for the testing and manipulation of single-bit flags contained within RAM digit locations.

**SKT** (SKip on Timer) instruction tests the state of an internal 10-bit time-base counter. This counter divides the instruction cycle clock frequency by 1024 and provides a latched indication of counter overflow. The SKT instruction tests this latch, executing the next program instruction if the latch is not set. If the latch has been set since the previous test, the next program instruction is skipped and the latch is reset. The features associated with this instruction, therefore, allow the controller to generate its own time-base for real-time processing rather than relying on an external input signal.

For example, using a 2.097 MHz crystal as the time-base to the clock generator, the instruction cycle clock frequency will be 131 kHz (crystal frequency ÷ 16) and the binary counter output pulse frequency will be 128 Hz. For time-of-day or similar real-time processing, the SKT instruction can call a routine which increments a "seconds" counter every 128 ticks.

### 3.3 COP421-Series Instruction Set Differences

The ININ instruction has been deleted. This is due to the lack of the IN inputs.

The INIL instruction has been substantially modified due to the lack of IN inputs and IL<sub>3</sub>/IL<sub>0</sub> latches. If an INIL instruction is executed on a COP421-series device, it will input only the state of CKO, providing CKO has been programmed as a general-purpose input (0 → A<sub>3</sub>, A<sub>1</sub>, A<sub>0</sub>; CKO → A<sub>2</sub>). If CKO has not been programmed as a general-purpose input, the INIL instruction is non-functional on the COP421-series.

### 3.4 COP410L/COP411L Instruction Set

The COP410L and COP411L instruction sets are subsets of the COP421-series instruction set.

Table 3.3 provides the mnemonic, operand, machine code, data flow, skip conditions and description associated with each instruction in the COP410L and COP411L instruction sets. An asterisk in the description column indicates the double-byte instruction. Notes are provided, following this

table, which include additional information relevant to particular instructions.

Table 3.4 provides a list of internal architecture, instruction operand and operational symbols used in the COP410L/COP411L Instruction Set Table. Table 3.7 provides an alphabetical mnemonic index of COP410L/COP411L instructions, indicating the hexadecimal opcode and description associated with each instruction. Table 3.8 is a list of COP410L/COP411L instructions arranged in order of their hexadecimal opcodes.

The following text discusses the differences which exist between the COP410L and COP411L instruction sets and that of the COP420-series. The COP410L is specifically discussed with differences between it and the COP411L noted. All other instructions perform the same machine operations and have the same typical usage as discussed in Section 3.2. For a treatment of the significance of those differences when writing programs for the COP410L and COP411L, see Section 3.5, COP410L/COP411L Instruction Set Differences, and Section 4.11, COP410L/COP411L Programming.

Table 3.3 COP410L/COP411L Instruction Set

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
ARITHMETIC INSTRUCTIONS						
ASC		30	<u>0 0 1 1</u>   <u>0 0 0 0</u>	A + C + RAM(B) → A Carry → C	Carry	Add with Carry, Skip on Carry
ADD		31	<u>0 0 1 1</u>   <u>0 0 0 1</u>	A + RAM(B) → A	None	Add RAM to A
AISC	y	5-	<u>0 1 0 1</u>   y	A + y → A	Carry	Add Immediate, Skip on Carry (y ≠ 0)
CLRA		00	<u>0 0 0 0</u>   <u>0 0 0 0</u>	0 → A	None	Clear A
COMP		40	<u>0 1 0 0</u>   <u>0 0 0 0</u>	$\bar{A}$ → A	None	Ones complement of A to A
NOP		44	<u>0 1 0 0</u>   <u>0 1 0 0</u>	None	None	No Operation
RC		32	<u>0 0 1 1</u>   <u>0 0 1 0</u>	"0" → C	None	Reset C
SC		22	<u>0 0 1 0</u>   <u>0 0 1 0</u>	"1" → C	None	Set C
XOR		02	<u>0 0 0 0</u>   <u>0 0 1 0</u>	A ⊕ RAM(B) → A	None	Exclusive-OR RAM with A

Table 3.3 COP410L/COP411L Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>TRANSFER OF CONTROL INSTRUCTIONS</b>						
JID		FF	11111111	ROM (PC <sub>8:A,M</sub> ) → PC <sub>7:0</sub>	None	Jump Indirect (Note 2)
JMP	a	6-	0110000 ag -- a7:0	a → PC	None	Jump
JP	a	--	1  a6:0 (pages 2,3 only) or 11  a5:0 (all other pages)	a → PC <sub>6:0</sub> a → PC <sub>5:0</sub>	None	Jump within Page (Note 3)
JSRP	a	--	10  a5:0	PC + 1 → SA → SB 010 → PC <sub>8:6</sub> a → PC <sub>5:0</sub>	None	Jump to Subroutine Page (Note 4)
JSR	a	6-	0110100 ag -- a7:0	PC + 1 → SA → SB a → PC	None	Jump to Subroutine
RET		48	01001000	SB → SA → PC	None	Return from Subroutine
RETSK		49	01001001	SB → SA → PC	Always Skip on Return	Return from Subroutine then Skip
<b>MEMORY REFERENCE INSTRUCTIONS</b>						
CAMQ		33 3C	001100011 00111100	A → Q <sub>7:4</sub> RAM(B) → Q <sub>3:0</sub>	None	Copy A, RAM to Q
LD	r	-5	00 r0101	RAM(B) → A Br ⊕ r → Br	None	Load RAM into A, Exclusive-OR Br with r
LQID		BF	10111111	ROM(PC <sub>8:A,M</sub> ) → Q SA → SB	None	Load Q Indirect (Note 2)
RMB	0 1 2 3	4C 45 42 43	01001100 01000101 01000010 01000011	0 → RAM(B) <sub>0</sub> 0 → RAM(B) <sub>1</sub> 0 → RAM(B) <sub>2</sub> 0 → RAM(B) <sub>3</sub>	None	Reset RAM Bit
SMB	0 1 2 3	4D 47 46 4B	01001101 01000111 01000110 01001011	1 → RAM(B) <sub>0</sub> 1 → RAM(B) <sub>1</sub> 1 → RAM(B) <sub>2</sub> 1 → RAM(B) <sub>3</sub>	None	Set RAM Bit
STII	y	7-	0111 y	y → RAM(B) Bd + 1 → Bd	None	Store Memory Immediate and Increment Bd
X	r	-6	00 r0110	RAM(B) ↔ A Br ⊕ r → Br	None	Exchange RAM with A, Exclusive-OR Br with r
XAD	3,15	23 BF	00100011 10111111	RAM(3,15) ↔ A	None	* Exchange A with RAM (3,15)
XDS	r	-7	00 r0111	RAM(B) ↔ A Bd - 1 → Bd Br ⊕ r → Br	Bd decrements past 0	Exchange RAM with A and Decrement Bd, Exclusive-OR Br with r
XIS	r	-4	00 r0100	RAM(B) ↔ A Bd + 1 → Bd Br ⊕ r → Br	Bd increments past 15	Exchange RAM with A and Increment Bd, Exclusive-OR Br with r

Table 3.3 COP410L/COP411L Instruction Set (continued)

Mnemonic	Operand	Hex Code	Machine Language Code (Binary)	Data Flow	Skip Conditions	Description
<b>REGISTER REFERENCE INSTRUCTIONS</b>						
CAB		50	$[0\ 1\ 0\ 1\  0\ 0\ 0\ 0]$	A → Bd	None	Copy A to Bd
CBA		4E	$[0\ 1\ 0\ 0\  1\ 1\ 1\ 0]$	Bd → A	None	Copy Bd to A
LBI	r,d	--	$[0\ 0\  r\  d-1]$ (d = 0, 9:15)	r,d → B	None	Load B Immediate with r,d (Note 5)
LEI	y	33 6-	$[0\ 0\ 1\ 1\  0\ 0\ 1\ 1]$ $[0\ 1\ 1\ 0\  y]$	y → EN	None	* Load EN Immediate (Note 6)
<b>TEST INSTRUCTIONS</b>						
SKC		20	$[0\ 0\ 1\ 0\  0\ 0\ 0\ 0]$		C = "1"	Skip if C is True
SKE		21	$[0\ 0\ 1\ 0\  0\ 0\ 0\ 1]$		A = RAM(B)	Skip if A Equals RAM
SKGZ		33 21	$[0\ 0\ 1\ 1\  0\ 0\ 1\ 1]$ $[0\ 0\ 1\ 0\  0\ 0\ 0\ 1]$		G <sub>3:0</sub> = 0	Skip if G is Zero (all 4 bits)
SKGBZ		33	$[0\ 0\ 1\ 1\  0\ 0\ 1\ 1]$	1st byte		* Skip if G Bit is Zero
	0	01	$[0\ 0\ 0\ 0\  0\ 0\ 0\ 1]$		G <sub>0</sub> = 0	
	1	11	$[0\ 0\ 0\ 1\  0\ 0\ 0\ 1]$		G <sub>1</sub> = 0	
	2	03	$[0\ 0\ 0\ 0\  0\ 0\ 0\ 1]$	2nd byte	G <sub>2</sub> = 0	
	3	13	$[0\ 0\ 0\ 1\  0\ 0\ 1\ 1]$		G <sub>3</sub> = 0	
SKMBZ		0 1 2 3	$[0\ 0\ 0\ 0\  0\ 0\ 0\ 1]$ $[0\ 0\ 0\ 1\  0\ 0\ 0\ 1]$ $[0\ 0\ 0\ 0\  0\ 0\ 0\ 1]$ $[0\ 0\ 0\ 1\  0\ 0\ 1\ 1]$		RAM(B) <sub>0</sub> = 0 RAM(B) <sub>1</sub> = 0 RAM(B) <sub>2</sub> = 0 RAM(B) <sub>3</sub> = 0	Skip if RAM Bit is Zero
<b>INPUT/OUTPUT INSTRUCTIONS</b>						
ING		33 2A	$[0\ 0\ 1\ 1\  0\ 0\ 1\ 1]$ $[0\ 0\ 1\ 0\  1\ 0\ 1\ 0]$	G → A	None	* Input G Ports to A
INL		33 2E	$[0\ 0\ 1\ 1\  0\ 0\ 1\ 1]$ $[0\ 0\ 1\ 0\  1\ 1\ 1\ 0]$	L <sub>7:4</sub> → RAM(B) L <sub>3:0</sub> → A	None	* Input L Ports to RAM, A
OBD		33 3E	$[0\ 0\ 1\ 1\  0\ 0\ 1\ 1]$ $[0\ 0\ 1\ 1\  1\ 1\ 1\ 0]$	Bd → D	None	* Output Bd to D Outputs
OMG		33 3A	$[0\ 0\ 1\ 1\  0\ 0\ 1\ 1]$ $[0\ 0\ 1\ 1\  1\ 0\ 1\ 0]$	RAM(B) → G	None	* Output RAM to G Ports
XAS		4F	$[0\ 1\ 0\ 0\  1\ 1\ 1\ 1]$	A ↔ SIO, C → SK	None	Exchange A with SIO (Note 2)

**Note 1:** All subscripts for alphabetical symbols indicate bit numbers unless explicitly defined (e.g., Br and Bd are explicitly defined). Bits are numbered 0 to N where 0 signifies the least significant (low-order, right-most bit). For example, A<sub>3</sub> indicates the most significant (left-most) bit of the 4-bit A register.

**Note 2:** For additional information on the operation of the XAS, JID, and LQID instructions, see Section 3.2.

**Note 3:** The JP instruction allows a jump, while in subroutine pages 2 or 3, to any ROM location within the two-page boundary of pages 2 or 3. The JP instruction, otherwise, permits a jump to a ROM location within the current 64-word page. JP may not jump to the last word of a page.

**Note 4:** A JSRP transfers program control to subroutine page 2 (0010 is loaded into the upper 4 bits of P). A JSRP may not be used when in pages 2 or 3. JSRP may not jump to the last word in page 2.

**Note 5:** LBI is a single-byte instruction if d = 0, 9, 10, 11, 12, 13, 14, or 15. The machine code for the lower 4 bits equals the binary value of the "d" data minus 7, e.g., to load the lower four bits of B (Bd) with the value 9 (1001<sub>2</sub>), the lower 4 bits of the LBI instruction equal 8 (1000<sub>2</sub>). To load 0, the lower 4 bits of the LBI instruction should equal 15 (1111<sub>2</sub>).

**Note 6:** Machine code for operand field y for LEI instruction should equal the binary value to be latched into EN, where a "1" or "0" in each bit of EN corresponds with the selection or deselection of a particular function associated with each bit. (See Functional Description, EN Register.)



Table 3.4 COP410L/411L Instruction Set Table Symbols

Symbol	Definition
INTERNAL ARCHITECTURE SYMBOLS	
A	4-bit Accumulator
B	6-bit RAM Address Register
Br	Upper 2 bits of B (register address)
Bd	Lower 4 bits of B (digit address)
C	1-bit Carry Register
D	4-bit Data Output Port
EN	4-bit Enable Register
G	4-bit Register to latch data for G I/O Port
L	8-bit TRI-STATE I/O Port
M	4-bit contents of RAM Memory pointed to by B Register
PC	9-bit ROM Address Register (program counter)
Q	8-bit Register to latch data for L I/O Port
SA	9-bit Subroutine Save Register A
SB	9-bit Subroutine Save Register B
SIO	4-bit Shift Register and Counter
SK	Logic-Controlled Clock Output

Symbol	Definition
INSTRUCTION OPERAND SYMBOLS	
d	4-bit Operand Field, 0-15 binary (RAM Digit Select)
r	2-bit Operand Field, 0-3 binary (RAM Register Select)
a	9-bit Operand Field, 0-511 binary (ROM Address)
y	4-bit Operand Field, 0-15 binary (Immediate Data)
RAM(s)	Contents of RAM location addressed by s
ROM(t)	Contents of ROM location addressed by t

OPERATIONAL SYMBOLS	
+	Plus
-	Minus
→	Replaces
↔	Is exchanged with
=	Is equal to
A	The ones complement of A
⊕	Exclusive-OR
:	Range of values

### 3.5 COP410L/COP411L Instruction Set Differences

#### Arithmetic Instructions

**ADT** has been deleted. To perform a similar operation an AISC 10 followed by a NOP to defeat the skip condition (carry) may be used.

**CASC** has been deleted. A COMP instruction followed by an ASC will achieve the same result (subtraction of A from M).

#### Input/Output Instructions

**ININ** has been deleted due to the COP410L's lack of IN inputs.

**OGI** has been deleted. A loading of data to the G ports must be accomplished via M by first loading M and then outputting its contents to G via an OMG instruction.

#### Memory Reference Instructions

**CQMA** has been deleted. Since no MICROBUST™ option is provided for the COP410L, Q is used in the COP410L primarily for output operations. An input of the L I/O ports, therefore, will effectively function as the equivalent of a CQMA; this is accomplished by the execution of an INL instruction.

**LDD** has been deleted. To load the contents of a data memory digit location into A, the usual procedure of loading B via an LBI to point to a particular RAM location followed by an LD instruction must be used.

**XAD** has been altered to reference *one* data memory location only; specifically, M(3,15). "Scratch-pad" data to be exchanged with A without affecting the B register should be placed, therefore, in M(3,15) and accessed by the XAD 3,15 instruction.

#### Register Reference Instructions

**LBI** has been altered to correspond to the data memory configuration of the COP410L. Specifically, it may only be used to access valid RAM locations, namely digits 9 through 15 and 0 in registers 0-3. The LBI "d" field, therefore, is limited to "d" values of 9-15 and 0, resulting in *all LBIs* being coded as *single-byte* instructions. Remember, the *machine code* for the "d" operand field is the binary value of "d" minus 1.

**XABR** has been deleted. To load Br, the entire B register must be loaded via an LBI. Altering Br may also be accomplished by using the EXCLUSIVE-OR "r" field associated with the memory reference instructions LD, X, XDS, and XIS.

#### Test Instructions

**SKT** has been deleted since the COP410L does not contain an internal divide-by-1024 time-base counter.

**Table 3.5 Alphabetical Mnemonic Index of COP420/COP421-Series Instructions**

Instruction	Hexadecimal Opcode	Description
ADD	31	ADD RAM to A
ADT	4A	ADd Ten to A
AISC 1-15	51-5F	Add Immediate, Skip on Carry
ASC	30	Add with carry, Skip on Carry
CAB	50	Copy A to Bd
CAMQ*	33/3C	Copy A, RAM to Q
CASC	10	Complement and Add with carry, Skip on Carry
CBA	4E	Copy Bd to A
CLRA	00	CLear A
COMP	40	COMPLement A
CQMA*	33/2C	Copy Q to RAM, A
ING*	33/2A	INput G ports to A
INIL*	33/29	INput IL latches to A**
ININ*	33/28	INput IN inputs to A**
INL*	33/2E	INput L ports to RAM, A
JID	FF	Jump InDirect
JMP*	60-63/00-FF	JuMP
JP	80-BE,CO-CE	Jump within Page
JSR*	68-6B/00-FF	Jump to SubRoutine
JSRP	80-BE	Jump to SubRoutine Page
LBI 0:9-15,0	08-0F	Load Bd Immediate (single-byte)
LBI 1:9-15,0	18-1F	
LBI 2:9-15,0	28-2F	
LBI 3:9-15,0	38-3F	Load Bd Immediate (double-byte)
LBI* 0:1-8	33/81-88	
LBI* 1:1-8	33/91-98	
LBI* 2:1-8	33/A1-A8	
LBI* 3:1-8	33/B1-B8	
LD 0,1,2,3	05,15,25,35	LoaD RAM into A
LDD* 0-3,0-15	23/00-3F	LoaD A with RAM, Directly
LEI* 0-15	33/60-6F	Load EN Immediate
LQID	BF	Load Q InDirect
NOP	44	No Operation
OBD*	33/3E	Output Bd to D outputs
OGI*	33/50-5F	Output to G ports Immediate
OMG*	33/3A	Output RAM to G ports
RC	32	Reset Carry
RET	48	RETurn
RETSK	49	RETurn then SKip
RMB 0,1,2,3	4C,45,42,43	Reset Memory Bit
SC	22	Set Carry
SMB 0,1,2,3	4D,47,46,4B	Set Memory Bit
SKC	20	SKip if Carry Is true
SKE	21	SKip if A Equals RAM
SKGBZ* 0,1,2,3	33/01,11,03,13	SKip if G Bit Is Zero
SKGZ*	33/21	SKip if G equals Zero (all 4 bits)
SKMBZ 0,1,2,3	01,11,03,13	SKip if Memory Bit Is Zero
SKT	41	SKip on Timer
STII	70-7F	STore memory Immediate and Increment Bd

**Table 3.5 Alphabetical Mnemonic Index of COP420/COP421-Series Instructions**

Instruction	Hexadecimal Opcode	Description
X 0,1,2,3	6,16,26,36	eXchange RAM with A, exclusive-OR r with Br
XABR	12	eXchange A with Br
XAD* 0-3,0-15	23/80-BF	eXchange A with RAM Directly
XAS	4F	eXchange A with SIO (serial I/O)
XDS 0,1,2,3	07,17,27,37	eXchange RAM with A and Decrement Bd
XIS 0,1,2,3	04,14,24,34	eXchange RAM with A and Increment Bd
XOR	02	eXclusive-OR RAM with A

\*Double-Byte Instruction: first byte/second byte (or first byte range/second byte range).

\*\*Instruction not available or has different features on COP421-series.

**Table 3.6 Table of COP420/COP421-Series Instructions Listed by Opcodes (Hexadecimal)**

00	CLRA	26	X 2
01	SKMBZ 0	27	XDS 2
02	XOR	28	LBI 2,9
03	SKMBZ 2	29	LBI 2,10
04	XIS 0	2A	LBI 2,11
05	LD 0	2B	LBI 2,12
06	X 0	2C	LBI 2,13
07	XDS 0	2D	LBI 2,14
08	LBI 0,9	2E	LBI 2,15
09	LBI 0,10	2F	LBI 2,0
0A	LBI 0,11	30	ASC
0B	LBI 0,12	31	ADD
0C	LBI 0,13	32	RC
0D	LBI 0,14	33	TWO WORD* (except LDD, XAD, JMP, JSR)
0E	LBI 0,15		
0F	LBI 0,0		
10	CASC	34	XIS 3
11	SKMBZ 1	35	LD 3
12	XABR	36	X 3
13	SKMBZ 3	37	XDS 3
14	XIS 0	38	LBI 3,9
15	LD 1	39	LBI 3,10
16	X 1	3A	LBI 3,11
17	XDS 1	3B	LBI 3,12
18	LBI 1,9	3C	LBI 3,13
19	LBI 1,10	3D	LBI 3,14
1A	LBI 1,11	3E	LBI 3,15
1B	LBI 1,12	3F	LBI 3,0
1C	LBI 1,13	40	COMP
1D	LBI 1,14	41	SKT
1E	LBI 1,15	42	RMB 2
1F	LBI 1,0	43	RMB 3
20	SKC	44	NOP
21	SKE	45	RMB 1
22	SC	46	SMB 2
23	LDD/XAD**	47	SMB 1
24	XIS 2	48	RET
25	LD 2	49	RETSK

Table 3.6 Table of COP420/COP421-Series Instructions  
Listed by Opcodes (Hexadecimal) (continued)

4A	ADT	7D	STII 13	6B	LEI 11	14	LDD 1,4
4B	SMB 3	7E	STII 14	6C	LEI 12	15	LDD 1,5
4C	RMB 0	7F	STII 15	6D	LEI 13	16	LDD 1,6
4D	SMB 0	80-BE	JP to word XX (0-3F <sub>16</sub> ) or JSRP to page 2, word XX (0-3F <sub>16</sub> ): opcode = 80 + XX	6E	LEI 14	17	LDD 1,7
4E	CBA			6F	LEI 15	18	LDD 1,8
4F	XAS			81	LBI 0,1	19	LDD 1,9
50	CAB			82	LBI 0,2	1A	LDD 1,10
51	AISC 1	BF	LQID	83	LBI 0,3	1B	LDD 1,11
52	AISC 2	C0-CE	JP to word XX (0-3F <sub>16</sub> ): opcode = C0 + XX	84	LBI 0,4	1C	LDD 1,12
53	AISC 3			85	LBI 0,5	1D	LDD 1,13
54	AISC 4			86	LBI 0,6	1E	LDD 1,14
55	AISC 5	FF	JID	87	LBI 0,7	1F	LDD 1,15
56	AISC 6			88	LBI 0,8	20	LDD 2,0
57	AISC 7		<b>Two Word Instructions, Second Word:</b>	91	LBI 1,1	21	LDD 2,1
58	AISC 8			92	LBI 1,2	22	LDD 2,2
59	AISC 9	*00	INIL (different features for COP421)	93	LBI 1,3	23	LDD 2,3
5A	AISC 10			94	LBI 1,4	24	LDD 2,4
5B	AISC 11	01	SKGBZ 0	95	LBI 1,5	25	LDD 2,5
5C	AISC 12	03	SKGBZ 2	96	LBI 1,6	26	LDD 2,6
5D	AISC 13	11	SKGBZ 1	97	LBI 1,7	27	LDD 2,7
5E	AISC 14	13	SKGBZ 3	98	LBI 1,8	28	LDD 2,8
5F	AISC 15	21	SKGZ	A1	LBI 2,1	29	LDD 2,9
60	JMP*** to Page 0, 1, 2, or 3	28	ININ (invalid for COP421)	A2	LBI 2,2	2A	LDD 2,10
61	JMP*** to Page 4, 5, 6, or 7	2A	ING	A3	LBI 2,3	2B	LDD 2,11
62	JMP*** to Page 8, 9, 10, or 11	2C	CQMA	A4	LBI 2,4	2C	LDD 2,12
63	JMP*** to Page 12, 13, 14, or 15	2E	INL	A5	LBI 2,5	2D	LDD 2,13
64	invalid	3A	OMG	A6	LBI 2,6	2E	LDD 2,14
65	invalid	3C	CAMQ	A7	LBI 2,7	2F	LDD 2,15
66	invalid	3E	OBD	A8	LBI 2,8	30	LDD 3,0
67	invalid	50	OGI 0	B1	LBI 3,1	31	LDD 3,1
68	JSR*** to Page 0, 1, 2, or 3	51	OGI 1	B2	LBI 3,2	32	LDD 3,2
69	JSR*** to Page 4, 5, 6, or 7	52	OGI 2	B3	LBI 3,3	33	LDD 3,3
6A	JSR*** to Page 8, 9, 10, or 11	53	OGI 3	B4	LBI 3,4	34	LDD 3,4
6B	JSR*** to Page 12, 13, 14, or 15	54	OGI 4	B5	LBI 3,5	35	LDD 3,5
6C	invalid	55	OGI 5	B6	LBI 3,6	36	LDD 3,6
6D	invalid	56	OGI 6	B7	LBI 3,7	37	LDD 3,7
6E	invalid	57	OGI 7	B8	LBI 3,8	38	LDD 3,8
6F	invalid	58	OGI 8	**00	LDD 0,0	39	LDD 3,9
70	STII 0	59	OGI 9	01	LDD 0,1	3A	LDD 3,10
71	STII 1	5A	OGI 10	02	LDD 0,2	3B	LDD 3,11
72	STII 2	5B	OGI 11	03	LDD 0,3	3C	LDD 3,12
73	STII 3	5C	OGI 12	04	LDD 0,4	3D	LDD 3,13
74	STII 4	5D	OGI 13	05	LDD 0,5	3E	LDD 3,14
75	STII 5	5E	OGI 14	06	LDD 0,6	3F	LDD 3,15
76	STII 6	5F	OGI 15	07	LDD 0,7	80	XAD 0,0
77	STII 7	60	LEI 0	08	LDD 0,8	81	XAD 0,1
78	STII 8	61	LEI 1	09	LDD 0,9	82	XAD 0,2
79	STII 9	62	LEI 2	0A	LDD 0,10	83	XAD 0,3
7A	STII 10	63	LEI 3	0B	LDD 0,11	84	XAD 0,4
7B	STII 11	64	LEI 4	0C	LDD 0,12	85	XAD 0,5
7C	STII 12	65	LEI 5	0D	LDD 0,13	86	XAD 0,6
		66	LEI 6	0E	LDD 0,14	87	XAD 0,7
		67	LEI 7	0F	LDD 0,15	88	XAD 0,8
		68	LEI 8	10	LDD 1,0	89	XAD 0,9
		69	LEI 9	11	LDD 1,1	8A	XAD 0,10
		6A	LEI 10	12	LDD 1,2	8B	XAD 0,11
				13	LDD 1,3		



**Table 3.6 Table of COP420/COP421-Series Instructions Listed by Opcodes (Hexadecimal) (continued)**

8C	XAD 0,12
8D	XAD 0,13
8E	XAD 0,14
8F	XAD 0,15
90	XAD 1,0
91	XAD 1,1
92	XAD 1,2
93	XAD 1,3
94	XAD 1,4
95	XAD 1,5
96	XAD 1,6
97	XAD 1,7
98	XAD 1,8
99	XAD 1,9
9A	XAD 1,10
9B	XAD 1,11
9C	XAD 1,12
9D	XAD 1,13
9E	XAD 1,14
9F	XAD 1,15
A0	XAD 2,0
A1	XAD 2,1
A2	XAD 2,2
A3	XAD 2,3
A4	XAD 2,4
A5	XAD 2,5
A6	XAD 2,6
A7	XAD 2,7
A8	XAD 2,8
A9	XAD 2,9
AA	XAD 2,10
AB	XAD 2,11
AC	XAD 2,12
AD	XAD 2,13
AE	XAD 2,14
AF	XAD 2,15
B0	XAD 3,0
B1	XAD 3,1
B2	XAD 3,2
B3	XAD 3,3
B4	XAD 3,4
B5	XAD 3,5
B6	XAD 3,6
B7	XAD 3,7
B8	XAD 3,8
B9	XAD 3,9
BA	XAD 3,10
BB	XAD 3,11
BC	XAD 3,12
BD	XAD 3,13
BE	XAD 3,14
BF	XAD 3,15

\*\*00+XX JSR or JMP to page 0, 4, 10, or 14, word XX (03F<sub>16</sub>): 0-3F  
 40+XX JSR or JMP to page 1, 5, 11, or 15, word XX (03F<sub>16</sub>):40-7F  
 80+XX JSR or JMP to page 2, 6, 12, or 16, word XX (03F<sub>16</sub>):80-BF  
 C0+XX JSR or JMP to page 3, 7, 13, or 17, word XX (03F<sub>16</sub>):C0-FF

**Table 3.7 Alphabetical Mnemonic Index of COP410L/COP411L-Series Instructions**

Instruction	Hexadecimal Opcode	Description
ADD	31	ADD RAM to A
AISC 1-15	51-5F	Add Immediate, Skip on Carry
ASC	30	Add with carry, Skip on Carry
CAB	50	Copy A to Bd
CAMQ	33/3C	Copy A, RAM to Q
CBA	4E	Copy Bd to A
CLRA	00	CLear A
COMP	40	COMPLement A
ING*	33/2A	INput G ports to A
INL*	33/2E	INput L ports to RAM, A
JID	FF	Jump INDirect
JMP*	60-61/00-FF	JuMP
JP	80-BE,C0-CE	Jump within Page
JSR*	68-69/00-FF	Jump to SubRoutine
JSRP	80-BE	Jump to SubRoutine Page
LBI 0;9-15,0	08-0F	Load Bd Immediate (single-byte)
LBI 1;9-15,0	18-1F	
LBI 2;9-15,0	28-2F	
LBI 3;9-15,0	38-3F	
LD 0,1,2,3	05,15,25,35	LoaD RAM Into A
LEI* 0-15	33/60-6F	Load EN Immediate
LQID	BF	Load Q INDirect
NOP	44	No OPeration
OBD*	33/3E	Output Bd to D outputs
OMG*	33/3A	Output RAM to G ports
RC	32	Reset Carry
RET	48	RETurn
RETSK	49	RETurn then SKip
RMB 0,1,2,3	4C,45,42,43	Reset Memory Bit
SC	22	Set Carry
SMB 0,1,2,3	4D,47,46,4B	Set Memory Bit
SKC	20	SKip if Carry is true
SKE	21	SKip if A Equals RAM
SKGBZ* 0,1,2,3	33/01,11,03,13	SKip if G Bit Is Zero
SKGZ*	33/21	SKip if G equals Zero (all 4 bits)
SKMBZ 0,1,2,3	01,11,03,13	SKip if Memory Bit Is Zero
STII	70-7F	STore memory Immediate and Increment Bd
X 0,1,2,3	6,16,26,36	eXchange RAM with A
XAD* 3,15	23-BF	eXchange A with RAM Directly
XAS	4F	eXchange A with SIO (serial I/O)
XDS 0,1,2,3	07,17,27,37	eXchange RAM with A and Decrement Bd
XIS 0,1,2,3	04,14,24,34	eXchange RAM with A and Increment Bd
XOR	02	eXclusive-OR RAM with A

\*Double-Byte Instruction: first byte/second byte (or first byte range/second byte range).

\*\*Instruction not available or has different features on COP421-series.

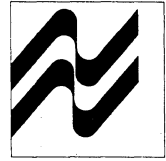
**Table 3.8 Table of COP410L/COP411L-Series Instructions  
Listed by Opcodes (Hexadecimal) (continued)**

00	CLRA	2E	LBI 2,15	5A	AISC 10	BF	LQID
01	SKMBZ 0	2F	LBI 2,0	5B	AISC 11	C0-CE	JP to word XX (0-3F <sub>16</sub> ): opcode = C0 + XX
02	XOR	30	ASC	5C	AISC 12	FF	JID
03	SKMBZ 2	31	ADD	5D	AISC 13		
04	XIS 0	32	RC	5E	AISC 14		
05	LD 0	33	TWO WORD* (except XAD, JMP, JSR)	5F	AISC 15		Two Word Instructions, Second Word:
06	X 0			60	JMP*** to Page 0, 1, 2, or 3	*00	invalid
07	XDS 0			61	JMP*** to Page 4, 5, 6, or 7	01	SKGBZ 0
08	LBI 0,9	34	XIS 3	64	invalid	03	SKGBZ 2
09	LBI 0,10	35	LD 3	65	invalid	11	SKGBZ 1
0A	LBI 0,11	36	X 3	66	invalid	13	SKGBZ 3
0B	LBI 0,12	37	XDS 3	67	invalid	21	SKGZ
0C	LBI 0,13	38	LBI 3,9	68	JSR*** to Page 0, 1, 2, or 3	28	invalid
0D	LBI 0,14	39	LBI 3,10	69	JSR*** to Page 4, 5, 6, or 7	2A	ING
0E	LBI 0,15	3A	LBI 3,11			2C	invalid
0F	LBI 0,0	3B	LBI 3,12	6C	invalid	2E	INL
10	invalid	3C	LBI 3,13	6D	invalid	3A	OMG
11	SKMBZ 1	3D	LBI 3,14	6E	invalid	3C	CAMQ
12	invalid	3E	LBI 3,15	6F	invalid	3E	OBD
13	SKMBZ 3	3F	LBI 3,0	70	STII 0	50-5F	invalid
14	XIS 0	40	COMP	71	STII 1	60	LEI 0
15	LD 1	41	invalid	72	STII 2	61	LEI 1
16	X 1	42	RMB 2	73	STII 3	62	LEI 2
17	XDS 1	43	RMB 3	74	STII 4	63	LEI 3
18	LBI 1,9	44	NOP	75	STII 5	64	LEI 4
19	LBI 1,10	45	RMB 1	76	STII 6	65	LEI 5
1A	LBI 1,11	46	SMB 2	77	STII 7	66	LEI 6
1B	LBI 1,12	47	SMB 1	78	STII 8	67	LEI 7
1C	LBI 1,13	48	RET	79	STII 9	68	LEI 8
1D	LBI 1,14	49	RETSK	7A	STII 10	69	LEI 9
1E	LBI 1,15	4A	invalid	7B	STII 11	6A	LEI 10
1F	LBI 1,0	4B	SMB 3	7C	STII 12	6B	LEI 11
20	SKC	4C	RMB 0	7D	STII 13	6C	LEI 12
21	SKE	4D	SMB 0	7E	STII 14	6D	LEI 13
22	SC	4E	CBA	7F	STII 15	6E	LEI 14
23	XAD**	4F	XAS			6F	LEI 15
24	XIS 2	50	CAB	80-BE	JP to word XX (0-3F <sub>16</sub> ) or JSRP to page 2, word XX (0-3F <sub>16</sub> ): opcode = 80 + XX	81-88	invalid
25	LD 2	51	AISC 1			91-98	invalid
26	X 2	52	AISC 2			A1-A8	invalid
27	XDS 2	53	AISC 3			B1-B8	invalid
28	LBI 2,9	54	AISC 4				
29	LBI 2,10	55	AISC 5				
2A	LBI 2,11	56	AISC 6				
2B	LBI 2,12	57	AISC 7				
2C	LBI 2,13	58	AISC 8				
2D	LBI 2,14	59	AISC 9				

\*\*00-BE Invalid BF -- XAD 3.15

\*\*\*00 + XX JSR or JMP to page 0 or 4 word XX (0-3F<sub>16</sub>): 0-3F  
40 + XX JSR or JMP to page 1 or 5 word XX (0-3F<sub>16</sub>): 40-7F  
80 + XX JSR or JMP to page 2 or 6 word XX (0-3F<sub>16</sub>): 80-BF  
C0 + XX JSR or JMP to page 3 or 7 word XX (0-3F<sub>16</sub>): C0-FF

# 4 COP400 Programming Techniques



This chapter provides several examples of programming techniques for COP400 devices. The COP420-series/COP444L instruction set is assumed since it falls between the smaller and larger instruction sets, respectively, of the COP410L and the COP440. For users of the COP410L/COP411L, Section 3.5 provides information on use of multiple COP410L instructions to simulate the function of COP420 instructions not provided for the COP410L. Users of the COP440 will find all examples relevant since this device contains all COP420 instructions as well as several additional instructions.

All examples are given in COPST<sup>TM</sup> Cross Assembler language, using COP400 assembler instruction mnemonics and operand statements. Although, in the following examples, instruction operands and ROM page numbers are written using decimal notation, the programmer may specify these expressions in hexadecimal notation — the assembler accepts either format (e.g., AISC 13 = AISC X'C, Page X'A = Page 10). On occasion, source code examples contain non-instruction statements, such as assembler directives which convey information to the assembler necessary for proper program address allocation and similar assembler related tasks. For further information on the COPS Cross Assembler and its use see *PDS User's Manual*, Chapter 8.

## 4.1 Program Memory Allocation

Generally, COP420-series program memory may be thought of as one area of 1024 bytes of ROM with an address range of 0 to 3FF (hexadecimal). However, while this concept is convenient in writing, assembling and debugging major portions of COP420-series programs, it is necessary, with respect to a few instructions, to conceptualize program memory on a 64-word "page" basis.

Specifically, because of the characteristics and restrictions associated with the JP, JSRP, JID, and LQID instructions, the programmer must conceive of program memory as 1024 bytes or words, organized as sixteen pages, numbered 0-15 respectively. The following discussion provides information and examples relating to the "page" characteristics of each of these unique instructions. For information on the machine code and operations performed by these instructions, see Section 3.2. Table 4.1 provides a conversion

chart indicating the hexadecimal address equivalents for each of the 16 "pages" of ROM. Note — each page consists of 0 through 3F<sub>16</sub> words.

Table 4.1 Page to Hexadecimal Address Table

Page	Hexadecimal Address Range
0	000-03F
1	040-07F
2	080-0BF
3	0C0-0FF
4	100-13F
5	140-17F
6	180-1BF
7	1C0-1FF
8	200-23F
9	240-27F
10	280-2BF
11	2C0-2FF
12	300-33F
13	340-37F
14	380-3BF
15	3C0-3FF

## JP Instruction

The JP instruction is used to transfer program control to a ROM location within a page or within a two-page boundary consisting of "subroutine pages" 2 or 3.

The following page restrictions apply to the JP instruction:

- When used in any page other than page 2 or 3, it can only jump to a word within the *current* page.
- When used in page 2 or 3, it may jump to a word within page 2 or 3.
- In all cases, it cannot jump to the last word of a page (word 03F<sub>16</sub>).

The JP instruction assembly operand normally consists of a program label or expression specifying the address of the word to be jumped to. To specify page boundaries and to ensure correct placement of the JP and other page-oriented



Thus, subroutines may share a common "return" subroutine, jumped to from page 2 or 3 with a single-byte JP instruction.

.PAGE 0		
JSRP	ADD	; CALL ADD SUBROUTINE
.PAGE 2		; START OF PAGE 2 CODE
ADD:		; ADD SUBROUTINE
JP	MEMOVE	; JUMP TO MEMOVE ; "RETURN" ROUTINE (NO ; "PUSH" OF STACK)
.PAGE 3		; START OF PAGE 3 CODE
MEMOVE:		; MEMORY MOVE ROUTINE
RET		; RETURN TO MAIN PROGRAM ; (POP STACK)

**JID Instruction**

The JID (Jump Indirect) instruction is another page-oriented instruction. For a machine operation description, see Section 3.2. JID is an *indirect* ROM addressing instruction which transfers program control to a new ROM location based upon the contents of a ROM "pointer." The paging features and restrictions associated with the JID instruction are as follows:

- JID first jumps to a ROM pointer based upon the contents of A and RAM.
- JID then transfers program control to the ROM word specified by the contents of the ROM pointer.
- The ROM pointer and the indirect address jumped to must be within the same 4-page ROM "block" as the JID instruction. Specifically, for purposes of this instruction, the sixteen pages of ROM are divided into 4 blocks as follows:

Block	Pages
1	0-3
2	4-7
3	8-11
4	12-15

For example, if the JID instruction is located in page 5, the ROM pointer and the indirect address to which program control is transferred must be within block 2 (pages 4-7). For an example of the use of the JID instruction in a simple keyboard decode routine, see Section 5.3.

**LQID Instruction**

The LQID instruction is an *indirect* data output instruction. It loads the 8-bit Q register with the

8-bit contents of a particular ROM location pointed to by A and RAM. For an explanation of the machine operations associated with this instruction, see Section 3.2. The paging restrictions associated with this instruction are similar to those associated with the JID instruction, as follows:

- For purposes of the LQID instruction as with the JID instruction, ROM is divided into 4-page ROM "blocks" (pages 0-3, 4-7, 8-11 and 12-15).
- The ROM location containing the LQID "lookup" data must be within the same ROM block as the LQID instruction.

For example, a LQID instruction located in page 9 must access ROM data located in pages 8 through 11.

**Additional Restrictions Associated with JP, JSRP, JID and LQID Instructions**

As already mentioned, the ROM address register (P) increments its value when executing an instruction to point to the next memory instruction, automatically "rolling over" to the next page after executing an instruction located in the last word of a page. It is important to realize, however, that P is incremented *prior to the execution of the current instruction*. This characteristic has important consequences for JP, JSRP, JID and LQID instructions which are located in the last word of a page. Specifically, these instructions will operate on the incremented value of P which, because of the increment-before-execution COP feature, will point to the first word of the next page. Consequently, if any of these instructions are placed in the last word of a page, the program will treat them as residing on the first word of the following page. Given the paging restrictions associated with these instructions, the following operations and restrictions are associated with the following placements of these instructions:

- A JP in the last word of a page will go to any location in the following page (except the last word). A JP in the last word of page 1 will be able to go to any location (except the last word) of page 2 or 3 since it is treated as a JP in page 2. Furthermore, a JP in the last word of page 3 will not go to a location within page 2 or 3, but, instead, will go to a location within page 4.
- A JSRP instruction is not allowed to reside in the last word of page 1, since it will be treated as an illegal use of JSRP in page 2. A JSRP in the last word of page 3, however, is allowed, since it will be treated as a JSRP outside of pages 2 or 3, namely in page 4.
- A LQID or JID instruction located in the last word of the last page of a particular ROM block (last word of page 3, 7, 11 or 15) will lookup data or transfer program control, respectively, to a location within the next 4 page ROM block.





The automatic data memory *digit* address increment and decrement features associated with the XIS and XDS instructions and their skip condition features facilitate the shifting, adding, and subtracting of the contents of data memory. Data that needs to be shifted should be located in adjacent digit locations (for example, the dotted-box locations in Figure 4.1). Data that needs to be added, subtracted, or shifted should be located in areas adjacent to the XIS or XDS skip boundaries. The dotted locations in Figure 4.1 are against the XIS boundary at digit 15. This allows the programmer to take advantage of the skip feature of the XIS instruction.

The following examples illustrate several of the principles discussed above. The notation  $M(N_1, N_2)$  indicates a particular data memory digit  $M$ , where  $N_1$  = register number and  $N_2$  = digit number.

```

; MOVE M(3,0) TO M(1,0)

LBI 3,0 ; 3 TO BR; 0 TO BD (SINGLE-BYTE
; LBI: D=0)
LD 2 ; M(3,0) TO A; 1 TO BR 3 @ 2 = 1)
X ; A TO M(1,0)

; MOVE MEMORY REGISTER 1 TO MEMORY REGISTER 0
; M(1,15) - M(1,0) TO M(0,15) - M(0,0)

LBI 1,15 ; 1 TO BR, 15 TO BD (SINGLE-BYTE
; LBI)
MV1: LD 1 ; M(1,15) TO A; 0 TO BR
XDS 1 ; A TO M(0,15); 1 TO BR; BD - 1 TO
; BD; CONTINUE TO MOVE NEXT
; LOWER DIGIT UNTIL BD GOES
; PAST 0 AND SKIPS
JP MV1 ; HERE IF NO SKIP

; LEFT SHIFT DOTTED AREAS OF FIGURE 4.1
; 0 TO M(0,12) → M(0,12) → M(0,13) → M(0,14) → M(0,15) TO A

CLRA ; 0 TO A
LBI 0,12 ; 0 TO BR; 12 TO BD
LSHFT XIS ; M(0,12) TO A; 0 TO M(0,12)
JP LSHFT ; EXCHANGE A INTO BD, LEFT
; SHIFT NEXT HIGHER DIGIT UNTIL
; "BD" GOES PAST 15 AND SKIPS

```

### 4.3 Subroutine Techniques

Any section of program code used repeatedly within the main program should be coded as a subroutine, preferably on "subroutine pages" 2 or 3 for the reasons discussed above. Subroutines are jumped to or "called" by the JSRP or JSR (double-byte) instruction, both of which "push" the stack, saving the next memory location address after the subroutine call in the SA subroutine-save register. The other subroutine-save registers are correspondingly pushed. Subroutine nesting on the COP420-series is permitted to 3 levels, since this device contains 3 subroutine-save registers.

Subroutines should terminate with a RET or RETSK instruction, both of which "pop" the subroutine stack, with the program return address in SA being placed in the program counter register. The other subroutine-save registers are also popped. The contents of SC, which is the bottom-most subroutine-save register, are retained in SC in addition to being placed in SB.

It is convenient to think of a subroutine as a program module. The programmer should make its interface to the calling program as clearly defined and as simple as possible. *The interface* (including data memory registers, entry points, etc., used by the subroutine) *should be documented fully by comments to the code.*

Subroutine examples presented in this chapter often use the double-byte JSR instruction to call subroutines since no restrictions are associated directly with its use. When writing an actual program, programmers should use the more efficient single-byte JSRP instruction as well as use the double-page boundaries of subroutine pages 2 and 3 for placement of subroutine code (as discussed above) for efficient single-byte jumps while in these pages using the JP instruction.

It is often useful to define multiple-entry points for a single subroutine. The successive-skip feature of the LBI instruction often facilitates this technique. For example, see Register Move Routines, Section 4.4.

The RETSK instruction allows the programmer to use an alternate return to the main program (skipping the first program instruction encountered upon return) based upon tests or computations made within the subroutine itself. Example:

```

.PAGE 0
.
.
.
JSRP ADD ; CALL ADD SUBROUTINE
; RETURN HERE IF RESULT ≤ 9
; RETURN HERE IF RESULT > 9
.
.
.PAGE 2 ; START PAGE 2 CODE
.
.
.
ADD: ADD ; ADD SUBROUTINE — ADDS TWO
; BCD DIGITS; RESULT TO A
.
.
AISC 7 ; OVERFLOW AND SKIP IF RESULT
; > 9
RET ; RETURN WITHOUT SKIP (RESULT
; < 9)
RETSK ; RETURN THEN SKIP (RESULT > 9)

```

## 4.4 Utility Routines

Programmers often build a library of basic routines which are useful in numerous applications. This and the following sections provide examples of several such "utility" routines.

### Register Move Routine

It is often necessary to move data from one memory register to another. The following are examples of this type of routine. Note that the routines may be easily modified to perform moves in the opposite direction (e.g., from register 1 to 0) or to include a move of register 1 to 2.

#### ADJACENT MEMORY MOVE ROUTINE

```

; ADJACENT MEMORY REGISTER MOVE, MULTIPLE ENTRY POINT SUBROUTINE
; MOV0T1: MOVE MEMORY REGISTER 0 TO REGISTER 1 ENTRY POINT
; MOV2T3: MOVE MEMORY REGISTER 2 TO REGISTER 3 ENTRY POINT
; ROUTINE MOVES DIGITS 15 THROUGH 0
; PREVIOUS CONTENTS OF A AND B ARE LOST

MOV0T1:  LBI        0,15      ; POINT TO M(0,15)
MOV2T3:  LBI        2,15      ; NOTE LBI SUCCESSIVE SKIP FEATURE
MOV:     LD         1         ; TRANSFER M TO A; EXCLUSIVE-OR 1 WITH BR
        XDS        1         ; EXCHANGE A WITH M; EXCLUSIVE-OR 1 WITH BR; DECREMENT BD
        JP         MOV       ; JUMP TO "MOV" IF MORE DIGITS TO MOVE
        RET        ; RETURN WHEN XDS SKIPS (LAST DIGIT MOVED)

```

#### DATA MEMORY SHIFT AND ROTATE ROUTINES

```

; MULTIPLE ENTRY POINT SUBROUTINE TO RIGHT SHIFT MEMORY REGISTER 0, 1, 2, OR 3 ONE DIGIT POSITION
; ZEROS ARE SHIFTED INTO DIGIT 15
; PREVIOUS CONTENTS OF A AND B ARE LOST
; RSH0: RIGHT SHIFT REGISTER 0 ENTRY POINT
; RSH1: RIGHT SHIFT REGISTER 1 ENTRY POINT
; RSH2: RIGHT SHIFT REGISTER 2 ENTRY POINT
; RSH3: RIGHT SHIFT REGISTER 3 ENTRY POINT

RSH0:    LBI        0,15      ; POINT TO DIGIT 15 IN APPROPRIATE REGISTER
RSH1:    LBI        1,15      ; NOTE LBI SUCCESSIVE SKIP FEATURE
RSH2:    LBI        2,15
RSH3:    LBI        3,15

SHFTR:   CLRA        ; ZEROS IN FIRST DIGIT (DIGIT 15)
        XDS        ; SHIFT RIGHT*
        JP         SHFTR     ; CONTINUE UNTIL ENTIRE REGISTER SHIFTED
        RET        ; RETURN WHEN FINISHED ("XDS" SKIPS)

```

\*NOTE THAT THE ABOVE ROUTINE CAN SHIFT THE REGISTERS ONE DIGIT TO THE *LEFT* USING THE "XIS" INSTRUCTION IN PLACE OF "XDS" AND STARTING AT DIGIT 0.

```

; MULTIPLE ENTRY POINT SUBROUTINE TO LEFT SHIFT THE BITS OF A MEMORY DIGIT
; UPON ENTRY, B MUST POINT TO THE DIGIT TO BE SHIFTED
; ZEROS ARE SHIFTED IN FROM THE RIGHT
; PREVIOUS CONTENTS OF A ARE LOST
; LEF1: SHIFT DIGIT LEFT 1 BIT ENTRY POINT
; LEF2: SHIFT DIGIT LEFT 2 BITS ENTRY POINT
; LEF3: SHIFT DIGIT LEFT 3 BITS ENTRY POINT

```

```

LEF3:    LD         ; DIGIT TO A
        ADD        ; ADD DIGIT TO ITSELF
        X         ; SHIFTED DIGIT TO MEMORY

LEF2:    LD
        ADD
        X

LEF1:    LD
        ADD
        X
        RET

```



```

; MULTIPLE ENTRY POINT SUBROUTINE TO LEFT ROTATE THE BITS OF A MEMORY DIGIT
; UPON ENTRY, B MUST POINT TO THE DIGIT TO BE ROTATED
; PREVIOUS CONTENTS OF A ARE LOST
; LR01: ROTATE DIGIT LEFT 1 BIT ENTRY POINT
; LR02: ROTATE DIGIT LEFT 2 BITS ENTRY POINT
; LR03: ROTATE DIGIT LEFT 3 BITS ENTRY POINT (SAME AS RIGHT ROTATE 1)

```

```

LOR3:   JSR      LR01      ; ROTATE 1, THEN 2 MORE
LOR2:   JSR      LR01
LOR1:   LD        ; DIGIT TO A
        ADD      ; ADD DIGIT TO ITSELF
        X        ; EXCHANGE M WITH A
        AISC     8      ; WAS MEMORY BIT3 ON?
        RET      ; NO, RETURN
        SMB      0      ; YES, WRAP AROUND BIT0
        RET

```

ACCUMULATOR SHIFT ROUTINE:

```

; SUBROUTINE TO LEFT SHIFT BITS OF A BY USING THE SIO REGISTER (SIO MUST BE ENABLED AS A SERIAL SHIFT REGISTER)
; SI MUST BE CONNECTED TO LOGIC "0" (GROUND)
; ZEROS ARE SHIFTED IN FROM THE RIGHT
; LFTA1: LEFT SHIFT A 1 BIT ENTRY POINT
; LFTA2: LEFT SHIFT A 2 BITS ENTRY POINT
; LFTA3: LEFT SHIFT A 3 BITS ENTRY POINT

```

```

LFTA1:  XAS              ; A TO SIO
LFTA2:  XAS              ; SIO TO A (SIO SHIFT RIGHT 1 BIT)
        RET
LFTA2:  XAS              ; A TO SIO
LFTA3:  JP      LFT2     ; DELAY 1 INSTRUCTION CYCLE TIME — SIO SHIFT RIGHT 1 MORE BIT
LFTA3:  XAS              ; A TO SIO
        JP      LFT3     ; DELAY 1 INSTRUCTION CYCLE TIME — SI SHIFT RIGHT 2 MORE BITS

```

CLEAR DATA MEMORY ROUTINE:

```

; SUBROUTINE TO CLEAR ALL RAM
; CLEAR REGISTERS 3 THROUGH 0 IN SUCCESSION, THEN RETURN

```

```

CLRAM:  LBI      3,15    ; START BY CLEARING REGISTER 3
CLR:    CLRA     ; 0 TO A
        XDS     CLR     ; EXCHANGE WITH DIGIT 15, DECREMENT DIGIT
        JP      CLR     ; CONTINUE UNTIL DIGIT 0 CLEARED
        XABR    ; BR TO A
        AISC     15     ; REGISTER 0 CLEARED?
        RET      ; YES, RETURN
        XABR    ; NO, REPLACE BR - 1 INTO BR
        JP      CLR     ; CLEAR NEXT REGISTER

```

4.5 Timing Considerations

Programmers must often synchronize programs with external events ("real-time" programming). Such programs must be balanced with respect to the execution times of the various branches taken by the program. To ensure equal execution times, program timing delays are added. There are numerous ways of introducing timing delays, the simplest but least efficient involving the use of NOPs. Obviously these are appropriate for only the shortest delays.

A counting loop, such as:

```

        CLRA     1
        AISC     1
        JP      -1      ; ADD 1 TO A UNTIL A
CONTINUE: ; OVERFLOWS*

```

is more efficient for longer delays, but destroys the previous contents of A. Another method is to use a "scratch-pad" counter in data memory using the XAD instruction. For example, assuming the use of a counter in M(3,15):

```

XAD  3,15  ; COUNTER TO A; A TO M(3,15)
AISC  1    ; ADD 1 TO COUNTER UNTIL IT
JP   -1    ; OVERFLOWS*
: XAD      ; RESTORE A THEN CONTINUE

```

\*Note: The above timing code example shows the use of a special assembler symbol in the operand of the JP instruction. Namely, the operand of the JP instruction, rather than using a program label, references the

assembler location counter (which equals the address of the current program address). The "." signifies the assembler location counter and the value of the operand equals the location counter minus the number of memory bytes to the right of the "." sign. Use of the "." location pointer symbol for transfer of control instructions facilitates coding in avoiding the need to create unique program labels to reference memory addresses.

Larger delays may be implemented by using multi-digit RAM counters. Another technique is calling unrelated subroutines which change registers or memory locations not currently in use or whose net effect on memory is null. An example of the latter technique is illustrated below.

```
JSR   LR03   ; LEFT ROTATE 3 BITS
JSR   LR01   ; LEFT ROTATE 1 MORE BIT
```

This combination of subroutines only affects A, while maintaining the integrity of data in the rotated memory digit.

#### 4.6 BCD Arithmetic Routines

BCD data manipulation routines are essential in applications which interface with human operators of a microcomputer system. They are easily

translated to and from codes used by decimal displays and keyboards. The COP400 series instruction set and internal architecture has been designed to perform BCD routines efficiently. The following routines are examples of simple BCD data manipulation routines.

##### Unsigned BCD Integer Add and Subtract Routines

The following programs present unsigned BCD integer add and subtract subroutines. Data is stored in data memory registers 0 and 1 and is 13 digits long, occupying memory digits 0 through 12, respectively. The most significant BCD digit is in memory digit 12. The techniques used to manipulate the contents of memory address register B are common to many arithmetic routines. The LD and XIS instructions transfer data between memory and A. After the transfer they modify B. LD 1 causes a "1" to be exclusive-ORed with Br. Since, in these routines, Br is always equal to 1 when the LD 1 instruction operates upon it, Br is always changed to 0. (LD 1 causes Br to point to memory register 0.) Similarly, XIS 1 also changes Br to point to memory register 0, *as well as* incrementing the value of Bd to point to the next higher memory *digit*. Thus, Br "flip-flops" between registers 1 and 0 while Bd "walks-up" the digits of the registers.

```
; SUBROUTINE TO DO UNSIGNED BCD INTEGER ADD OF R1 AND R0, RESULT TO R0
; EACH INTEGER OCCUPIES MEMORY DIGITS 0 (LOW ORDER) THROUGH 12 (HIGH ORDER)
; ON RETURN, C = 1 INDICATES OVERFLOW
; PREVIOUS CONTENTS OF A AND B ARE LOST
; ENTRY POINT: BCDADD
```

```
BCDADD:  LBI     1,0           ; POINT TO LOW ORDER DIGIT, REGISTER 1
          RC          ; INITIALIZE C TO "0" (NO CARRY)
ADDL:    LD      1           ; MOVE R1 DIGIT TO A, POINT TO SAME DIGIT IN R0
          AISC     6           ; ADD BCD CORRECTION FACTOR OF 6 TO A
          ASC      ; ADD R0 DIGIT TO R1 DIGIT
          ADT      ; RESTORE BCD VALUE IF BCD CORRECTION NOT NECESSARY
          XIS     1           ; MOVE SUM DIGIT TO R0: POINT TO R1, NEXT HIGHER DIGIT
          CBA     ; BD TO A
          AISC     3           ; LAST DIGITS ADDED?
          JP      ADDL        ; NO, ADD NEXT HIGHER DIGITS
          RET          ; YES, RETURN
```

```
; SUBROUTINE TO DO UNSIGNED BCD INTEGER SUBTRACT
; MINUEND IS IN R0, SUBTRAHEND IS IN R1
; DIFFERENCE IS PLACED IN R0
; MINUEND, SUBTRAHEND AND DIFFERENCE DIGITS EACH OCCUPY MEMORY DIGITS 0 (LOW ORDER) THROUGH 12 (HIGH ORDER)
; ON RETURN: C = 1 INDICATES NO BORROW, C = 0 INDICATES BORROW
; PREVIOUS CONTENTS OF A AND B ARE LOST
; ENTRY POINT: BCDSUB
```

```
BCDSUB:  LBI     1,0           ; POINT TO LOW ORDER DIGIT IN R1
          SC          ; INITIALIZE C TO "1" (NO BORROW)
SUB:     LD      1           ; LOAD R1 DIGIT TO A, POINT TO SAME DIGIT IN R0
          CASC     ; SUBTRACT R1 DIGIT FROM R0 DIGIT
          ADT      ; BCD ADJUST IF BORROW (C = 0)
          XIS     1           ; PLACE DIFFERENCE DIGIT IN R0, POINT TO NEXT HIGHER DIGIT IN R1
          CBA     ; BD TO A
          AISC     3           ; HIGH ORDER DIGITS (12) SUBTRACTED?
          JP      SUB        ; NO, SUBTRACT NEXT HIGHER DIGITS
          RET          ; YES, RETURN
```

### BCD Integer Multiply Routine

This routine will multiply the contents of data memory register 2 with register 1, placing the result in register 2 (digits 0-12). It also calls the BCD add routine ("BCDADD") given above. Note that a loop-counter is contained in M(0,13) which causes the program to return after all 12 digits have been multiplied. Also note the alternate-return feature of page 3 subroutine TMZERO (Test Memory Digit = 0). A flowchart for the routine is given in Figure 4.2.

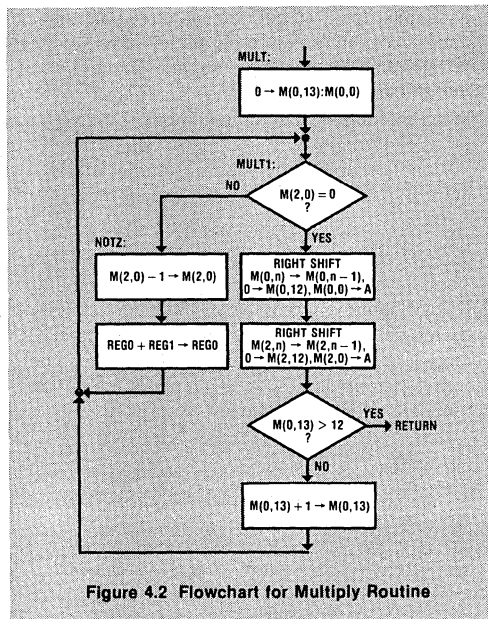


Figure 4.2 Flowchart for Multiply Routine

```

; TWO-LEVEL BCD INTEGER MULTIPLY SUBROUTINE
; 12 DIGIT BCD INTEGER CONTAINED IN REGISTER 1, DIGITS 0 - 12 (LOW ORDER TO HIGH ORDER) MULTIPLIED BY 12 DIGIT BCD
; INTEGER CONTAINED IN REGISTER 2, DIGITS 0 - 12 (LOW ORDER TO HIGH ORDER), RESULT TO REGISTER 2
; MULTIPLICATION OF DIGITS PERFORMED BY MULTIPLE ADDITIONS OF REGISTER 1 ACCORDING TO VALUE OF REGISTER 2
; DIGITS
; DIGIT ADDITION RESULTS TEMPORARILY STORED IN R0 AND CONSECUTIVELY RIGHT SHIFTED INTO RESULT REGISTER 2, HIGH
; ORDER DIGIT
; ENTRY POINT: MULT
; SUBROUTINES CALLED: RSHR0, RSHR2, CLR, DEC 1, INC 1, TMZERO, BCDADD
  
```

```

MULT:   LBI      0,13      ; POINT TO M(0,13)
        JSR      CLR      ; CLEAR REGISTER 0, DIGITS 13 - 0
MULT1:  LBI      2,0      ; POINT TO M(2,0)
        JSR      TMZERO   ; IS M(2,0) = 0?
        JP       NOTZ     ; NO, JUMP TO NOTZ
        JSR      RSHR0    ; YES, RIGHT SHIFT REGISTER 0, DIGITS 12 - 0
        JSR      RSHR2    ; RIGHT SHIFT REGISTER 2, DIGITS 12 - 0
        LBI      0,13     ; POINT TO LOOP COUNTER
        LD       A,A      ; LOOP COUNTER TO A
        AISC     3        ; IS COUNTER > 12
        JP       .+2     ; NO, CONTINUE
        RET        ; YES, ALL DIGITS MULTIPLIED, RETURN
        JSR      INC1     ; CONTINUE, INCREMENT LOOP COUNTER DIGIT
        JP       MULT1    ; MULTIPLY NEXT HIGHER ORDER DIGITS
NOTZ:   JSR      DEC1     ; DECREMENT M(2,0)
        JSR      BCDADD   ; ADD R0, DIGITS 0 - 12, TO R1, DIGITS 0 - 12, RESULT TO R0
        JP       MULT1    ; JUMP BACK TO MULT 1
  
```

```

; MULTIPLE ENTRY POINT SUBROUTINE TO RIGHT SHIFT DIGITS 12 - 0 OF REGISTER 0 OR 2
; ON RETURN A CONTAINS LOW ORDER REGISTER DIGIT
; RSHR0: RIGHT SHIFT DIGITS OF REGISTER 0 ENTRY POINT
; RSHR2: RIGHT SHIFT DIGITS OF REGISTER 2 ENTRY POINT
  
```

```

RSHR0:  LBI      0,12     ; POINT TO HIGH ORDER DIGIT, REGISTER 0
RSHR2:  LBI      2,12     ; POINT TO HIGH ORDER DIGIT, REGISTER 2
RSH:    XDS      RSH      ; SHIFT RIGHT DIGITS 12 - 0 IN REGISTER
        JP       RET
  
```

```
; SUBROUTINE TO CLEAR ALL DIGITS TO THE RIGHT AND INCLUSIVE OF A HIGH-ORDER DIGIT OF A REGISTER
; ON ENTRY, B MUST POINT TO THE REGISTER AND HIGH ORDER DIGIT NUMBER
```

```
CLR:      CLRA
          XDS                ; CLEAR REGISTER, STARTING WITH HIGH ORDER DIGIT
          JP      CLR        ; RETURN WHEN DIGIT 0 CLEARED
          RET
```

```
; MULTIPLE ENTRY SUBROUTINE TO EITHER DECREMENT OR INCREMENT BY 1 THE VALUE OF A MEMORY DIGIT
; ON ENTRY, B MUST POINT TO THE DIGIT TO BE OPERATED UPON
; DEC1: ENTRY POINT TO DECREMENT A DIGIT
; INC1: ENTRY POINT TO INCREMENT A DIGIT
```

```
DEC1:    CLRA
          COMP                ; 15 TO A
ADEX:    ADD                ; ADD MEMORY DIGIT TO A
          X                  ; EXCHANGE BACK TO MEMORY
          RET                ; RETURN
INC1:    CLRA
          AISC      1        ; 1 TO A
          JP      ADEX      ; ADD AND EXCHANGE WITH MEMORY DIGIT
```

```
; SUBROUTINE TO TEST MEMORY DIGIT EQUAL TO ZERO
; ON ENTRY, B MUST POINT TO MEMORY DIGIT TO BE TESTED
; ON RETURN, SKIP FIRST INSTRUCTION IF MEMORY DIGIT EQUAL TO ZERO
; NORMAL RETURN IF MEMORY DIGIT NOT EQUAL TO ZERO
```

```
TMZERO:  CLRA                ; 0 TO A
          SKE                ; DIGIT = ZERO?
          RET                ; NO, NORMAL RETURN
          RETSK              ; YES, RETURN THEN SKIP
```

## 4.7 Simple Display Loop Routine

The following routine is a simple LED display loop routine. It illustrates the use of LEI and LQID instructions, both designed to facilitate the outputting of segment data to a multiplexed display. As explained in Section 3.2, LEI Instruction description, setting bit 2 of the EN register enables Q latch (segment) data to the L I/O ports; resetting EN<sub>2</sub> disables the L I/O ports, providing segment blanking for the LED display. EN<sub>2</sub> is set and reset, respectively, by the LEI 4 and LEI 0 instructions.

As explained in Sections 3.2 and 4.1, LQID loads the 8-bit Q register with the contents of a ROM location pointed to by A and M (ROM "lookup" data must be within the same 4-page ROM block as the LQID instruction). In this example, since A is always equal to 0 at the time of the LQID instruction, the ROM data accessed by this instruction must be within the first 16 words of the first page of the ROM block in which the LQID instruction is located as pointed to by the 4-bit contents of M (P<sub>9</sub> and P<sub>8</sub> remain the same, P<sub>7</sub>-P<sub>4</sub> equal "0"). For example, if, as is the case for the following routine, LQID is in page 5, it will lookup data within one of the first 16 locations of page 4. The value of the contents of the memory digit pointed to by the B register at the time of the LQID instruction determines which one of the 16 words is accessed (e.g., if M = 2, word 2 is loaded into Q).

Due to these considerations, page 4, words 0-9 should equal the 8-bit, seven-segment decode lookup data for the BCD digits 0-9 respectively. (In this example the low-order bit — decimal point — of each lookup data word is reset, signifying that the decimal point is off.) ROM seven-segment decode lookup data is placed in ROM memory locations by the Assembler .WORD directive. (See *PDS User's Manual*, Section 8.4.)

Another feature of this routine is the dual function of Bd. Its value may be output directly to the D outputs to select one of 16 digits of the multiplexed display (assuming the D outputs are connected to a 1-of-16 decoder/driver device). Also, its value is used to select one of 16 RAM digits whose contents are used by the LQID instruction to access the segment data to be output to the selected digit. To facilitate coding (by avoiding the need to change the value of Bd after its contents are output to D to select or display digit), RAM digit locations should correspond to the digit of the display. In other words, RAM digits 0-15 should contain, respectively, the LQID pointers to segment data for display digits 0-15. This technique, used below, allows Bd to first enable the appropriate display digit and then, without its value being changed, to point to the RAM digit used to access the segment data for the same display digit.

```
; SEVEN-SEGMENT DECODE DATA TABLE:
; ROM BITS I7 - I0 = SA - SG, D.P. (DECIMAL POINT) BITS, RESPECTIVELY
```

```
LOOKUP: .PAGE      4          ; PLACE LOOKUP DATA IN WORDS 0 - 9, PAGE 4
        .WORD     X'FC      ; = 0 (SEVEN-SEGMENT DECODE HEX VALUES)
        .WORD     X'60      ; = 1
        .WORD     X'DA      ; = 2
        .WORD     X'F2      ; = 3
        .WORD     X'66      ; = 4
        .WORD     X'B6      ; = 5
        .WORD     X'BE      ; = 6
        .WORD     X'E0      ; = 7
        .WORD     X'F4      ; = 8
        .WORD     X'F6      ; = 9
        .          ; NEXT FIVE LOCATIONS CAN BE USED FOR SPECIAL ALPHABETICAL DISPLAY
        .          ; CHARACTER DATA
```

```
; BEGIN CODE FOR DISPLAY LOOP
```

```
DSPLY:  .PAGE      5          ; PLACE FOLLOWING CODE ON PAGE 5
        LBI       0,15      ; POINT TO HIGH ORDER RAM DIGIT, BD = 15
LOOP:   CLRA       ; A = 0 FOR LOOKUP
        LEI       0        ; BLANK SEGMENTS (EN2 = 0)
        OBD       ; OUTPUT DIGIT VALUE
        LQID      ; LOOKUP DATA TO Q
        LEI       4        ; OUTPUT SEGMENT DATA (EN2 = 1)
        CBA       ; BD TO A
        AISC      15       ; DECREMENT A
        JP        .+3      ; JUMP 3 WORDS WHEN FINISHED
        CAB       ; A(BD - 1) TO BD
        JP        LOOP     ; DISPLAY NEXT LOWER DIGIT
        .          ; CONTINUE WHEN FINISHED
```

## 4.8 Interrupt Service Routine

As explained in Section 3.2, LEI Instruction description, setting bit 1 of the EN register enables the COP420-series and COP444L IN<sub>1</sub> input as an interrupt input, responding to low going pulses. Upon the occurrence of an interrupt signal, the subroutine stack is pushed and program control is transferred to the last word of page 3 (address OFF<sub>16</sub>). The following routine contains code which may be placed at the beginning and end of the interrupt service routine to save the contents of A, C and B, freeing them for use by the interrupt routine. At the end of the routine the previous contents of A, C and B are restored for use by the main program. It should be noted that the main program need only enable IN<sub>1</sub> as an interrupt input once; thereafter, the interrupt service routine, itself, re-enables interrupt servicing (LEI 1 instruction before return).

```

; INTERRUPT SERVICE ROUTINE TO SAVE AND RESTORE THE CONTENTS OF A, C AND B (BR AND BD) IN MEMORY REGISTER 0,
; DIGITS 0 - 2.
; AUTOMATIC ENTRY TO LAST WORD OF PAGE 3
; ON RETURN, IN1 INPUT RE-ENABLED AS INTERRUPT INPUT

INTSER:  NOP                ; FIRST INTERRUPT ROUTINE INSTRUCTION MUST BE A NOP (LOCATION X'FF)
        XAD      0,0        ; SAVE A IN M(0,0)
        CBA                ; BD TO A
        XAD      0,1        ; SAVE BD IN M(0,1)
        XABR                ; BR TO A
        SKC                ; CARRY = 1?
        AISC      8         ; NO, SET A3
        XAD      0,2        ; SAVE C AND BR IN M(0,2)
        .                ; PERFORM INTERRUPT ROUTINE
        .
        LDD      0,2        ; M(0,2) (C AND BR) TO A
        RC                ; RESET CARRY
        AISC      8         ; A3 SET (SAVED CARRY = 0)?
        SC                ; NO, RESTORE CARRY = 1
        XABR                ; RESTORE BR
        LDD      0,1        ; M(0,1) (BD) TO A
        CAB                ; RESTORE BD
        LDD      0,0        ; M(0,0) TO A, RESTORE A
        LEI      1         ; ENABLE INTERRUPT (SET IN1)
        RET                ; RETURN FROM INTERRUPT SERVICE ROUTINE

```

## 4.9 Timekeeping Routine

The following multilevel subroutine counts time in a 12-hour format. It relies on the COP420 system oscillator, itself (controlled by an inexpensive 3.58 MHz color TV crystal), and the COP420 internal time-base counter for a real-time base, rather than on a 60Hz external input. The subroutine is entered each time the SKT instruction skips, indicating time-base counter overflow. As explained in Section 3.2, SKT Instruction description, overflow frequency is dependent upon the frequency of the COPST<sup>TM</sup> system oscillator. This frequency equals the oscillator frequency, first divided by 16 by the instruction cycle divider, then by 1024 by the internal 10-bit time-base counter. In this case the SKT overflow frequency will equal a fractional

number: 218.478 Hz (3.58 MHz divided by 16, divided by 1024). Consequently, the timekeeping *calling* routine must execute a SKT instruction at least once approximately each 218 Hz to ensure that each SKT overflow is detected.

As indicated above, using an inexpensive TV crystal results in a fractional SKT frequency. Program compensation techniques, therefore, must be employed to derive an integer which may be used by the program in counting seconds, the basic timekeeping units.

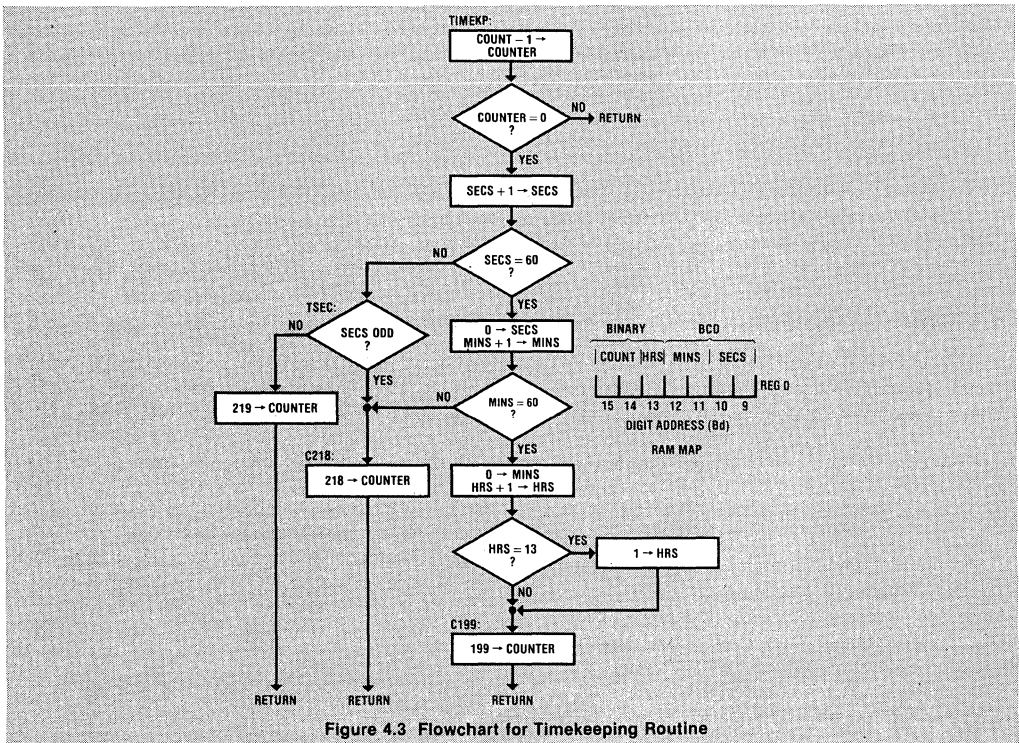
This routine derives this integer and utilizes it to keep accurate time in the following manner:

- A 2-digit binary "SKT" counter in RAM is initialized to different values at different times during the course of an hour so that the total counts for the hour equal an integer which corresponds to the 218.478 Hz SKT frequency.
- Every odd second in the range of 0-59 seconds, the SKT counter is set to 218, decremented by 1 each time the SKT instruction skips. When decremented to 0, a 2-digit BCD "seconds" counter in RAM is incremented by 1. (The seconds counter overflows every 60 counts to a 2-digit BCD "minute" counter. The minutes counter overflows every 60 counts to a 1-digit "hours" counter.)
- Every even second in the range of 0-59 seconds, the SKT counter is set to 219 and decremented by 1, as above, each time the SKT pulse occurs.
- Every minute in the range of 0-59 minutes, the SKT counter is set to 218 and decremented as above.
- Every hour, the SKT counter is set to 199 and decremented as above.

The above compensation techniques result in a timekeeping routine which is accurate at the end of each hour. (During the hour, inaccuracy is extremely small.) The basis for the above compensation scheme is as follows:

- Using a 3.58 MHz crystal resulting in a 218.478 Hz SKT frequency, an SKT integer count of 786,521 is obtained each hour ( $218.478 \times 3600$  seconds/hour).
- Using the above compensation scheme, the same number of "SKT" counts (786,521) is required to increment the time by 1 hour. This follows since 392,400 counts are required by the "odd" seconds compensation ( $30 \times 60 \times 218$  counts); 381,060 by the "even" seconds compensation ( $29 \times 60 \times 219$  counts); 12,862 by the "minutes" compensation ( $59 \times 218$  counts) and 199 by the "hours" compensation — resulting in a total hours count of 786,521.

A flowchart and a RAM map for this routine are provided in Figure 4.3. Note that an assembler assignment statement is used in the assembler source code to equate the address of low order digits of the RAM SKT counter and seconds counter with the symbols "COUNT" and "SECS," respectively. This provides clearer documentation of the program since an instruction referencing the seconds counter, for instance, can use the word "SECS" instead of a numerical value in the operand field (i.e., LBI SECS). For further information on the assignment statement, see *PDS User's Manual*, Section 8.4. Also note that the program initializes the SKT counter to 218, 219 and 199, respectively, by loading its two digits with the following binary equivalent pairs (high-order value, low-order value): 13, 10; 13, 11; and 12, 7.



This subroutine is coded to reside on subroutine page 2. The source code provided below also illustrates the use of the PDS Assembler .LOCAL directive and local symbol labels. Specifically, the program begins and ends with a .LOCAL directive, making the memory addresses between them a local region. Within this local region, local symbols (labels whose first character is a "\$") will be defined only within the local region — they will not conflict with labels appearing in other portions of program source code. This relieves the programmer from worry about duplicate label definitions, allowing the subroutine or other utility program to be included or added to different programs, regardless of the labels used by these other programs.

In effect, therefore, utility programs or commonly used subroutines may be coded in this manner and

placed in separate "utility" files on a disk. They can then be added or included, when needed, to main programs at a later date. For an example of a program which includes this "TIMEKP" subroutine (using the assembler .INCLD directive), see Figure 5.18.

Local symbols must begin with a "\$" and be unique within the particular local region in the first 4 characters following the "\$." The programmer may, as is done in this example, use local labels with more than four characters for convenience and, although not "recognized" by the assembler, these extra characters will be printed out on the assembler output listing. Note: The label of the starting address of a local utility routine must be a *long* (regular) label, since it will be referenced by a portion of the program outside of the local region (e.g., "TIMEKP" is not a local label).

```
; PAGE 2 SUBROUTINE TO KEEP TIME IN A 12-HOUR FORMAT USING A 3.58 MHZ TV CRYSTAL
; 2-DIGIT "SKT" COUNTER CONTAINED IN M(2,15) - M(2,14): HIGH- TO LOW-ORDER
; 1-DIGIT BINARY HOURS COUNTER IN M(2,13)
; 2-DIGIT BCD MINUTES COUNTER IN M(2,12) - M(2,11): HIGH- TO LOW-ORDER
; 2-DIGIT BCD SECONDS COUNTER IN M(2,10) - M(2,9): HIGH- TO LOW-ORDER
; ENTRY POINT: TIMEKP; ENTRY UPON SKT INSTRUCTION OVERFLOW
; SUBROUTINES CALLED: INC2
```

```

.PAGE          2          ; PAGE 2 SUBROUTINE
.LOCAL         ; CREATE LOCAL REGION FOR LOCAL SYMBOLS
$COUNT      = 2,14      ; ASSIGN "COUNT" = ADDRESS OF LOW-ORDER SKT COUNTER DIGIT
$SECS        = 2,9      ; ASSIGN "SECS" = ADDRESS OF LOW-ORDER SECONDS COUNTER DIGIT

TIMEKP:
LBI           $COUNT    ; POINT TO LOW-ORDER DIGIT OF SKT COUNTER
LD            ; LOAD DIGIT TO A
AISC         15          ; DIGIT = 0? (A = DIGIT - 1)
JP           $HIGHST     ; YES, TEST HIGH-ORDER DIGIT
X            ; NO, EXCHANGE DIGIT - 1 INTO M
RET          ; RETURN UNTIL NEXT SKT OVERFLOW
$HIGHST:
XIS          ; REPLACE DIGIT IN COUNTER, INCREMENT BD
JP           TIMEKP + 1  ; JUMP BACK AND TEST HIGH-ORDER DIGIT — IF ALREADY TESTED AND = 0.
                ; SKIP AND CONTINUE
LBI          $SECS       ; POINT TO LOW-ORDER SECS DIGIT
JSR         $INC2        ; INCREMENT SECS COUNTER
JP          $TSEC        ; SECS < 60, TEST SECS FOR ODD OR EVEN
STII        0            ; SECS = 60, 0 TO HIGH-ORDER DIGIT, POINT TO LOW-ORDER MINS DIGIT
JSR         $INC2        ; INCREMENT MINS COUNTER
JP          $C218        ; MINS < 60, SET COUNTER = 218
STII        0            ; MINS = 60, 0 TO HIGH-ORDER DIGIT, POINT TO HOURS DIGIT
LD            ; LOAD HOURS DIGIT TO A
AISC         1          ; INCREMENT HOURS
X            ; PLACE IN M, PREVIOUS HRS TO A
AISC         4          ; HOURS > 12?
JP          $C199        ; NO, SET COUNTER = 199
STII        1            ; YES, SET HOURS = 1
$C199:
LBI          $COUNT    ; POINT TO LOW-ORDER COUNTER DIGIT
STII        7            ; SET COUNTER = 199 (BINARY 12,7)
STII        12          ;
RET          ; RETURN UNTIL NEXT SKT OVERFLOW
$TSEC:
LBI          $SECS       ; POINT TO LOW-ORDER SECS DIGIT
SKMBZ       0            ; SECS ODD?
JP          $C218        ; YES, SET COUNTER = 218 (BINARY 13,10)
$C219:
LBI          $COUNT    ; NO, POINT TO LOW-ORDER COUNTER DIGIT
STII        11          ; SET COUNTER = 219 (BINARY 13,11)
$C21X
STIII13
RET
$C218:
LBI          COUNT      ; POINT TO LOW-ORDER COUNTER DIGIT
STII        10          ; SET COUNTER = 218
JP          $C21X       ; JUMP TO "C21X" THEN RETURN
```



```

; SUBROUTINE TO INCREMENT A 2-DIGIT BCD RAM COUNTER
; ON ENTRY, B MUST POINT TO LOW-ORDER DIGIT OF COUNTER
; ENTRY POINT: INC2
; NORMAL RETURN IF 2-DIGIT VALUE LESS THAN 60
; RETURN THEN SKIP IF 2-DIGIT VALUE EQUAL TO 60
; BOTH RETURNS EXIT WITH B POINTING TO HIGH-ORDER DIGIT

```

```

SINC2:
    SC                ; INITIALIZE C TO 1 TO ADD TO LOW-ORDER DIGIT
    CLRA             ; ZERO TO A
    AISC             6 ; BCD ADJUST RESULT IF NECESSARY
    ASC              ; IF RESULT > 9, LOW ORDER DIGIT = 0
    ADT
    XIS              ; PLACE INCREMENTED DIGIT IN M, POINT TO HIGH-ORDER DIGIT
    CLRA             ; ZERO TO A
    AISC             6 ; ADD CARRY, IF PROPAGATED FROM LOW-ORDER DIGIT TO HIGH-ORDER DIGIT
    ASC
    ADT              ; BCD RESULT IF NECESSARY
    X                ; REPLACE DIGIT IN M
    LD               ; LOAD HIGH-ORDER DIGIT INTO A
    AISC             10 ; HIGH-ORDER DIGIT = 6 (COUNT = 60)?
    RET              ; NO, NORMAL RETURN
    RETSK            ; YES, RETURN THEN SKIP
    .LOCAL           ; END LOCAL REGION

```

### 4.10 String Search Routine

It is often necessary to search data memory for a string of characters. The following routine searches register 0 for a match with three contiguous 4-bit characters, "X," "Y," and "Z." Note that a match with more than three characters is easily accommodated by providing for additional

character tests, using the simple character test instructions provided below containing modified LDD instructions whose operands specify the additional characters to be matched. Also, the code may be easily modified to search through more than one RAM register for a match.

```

; SUBROUTINE TO SEARCH STRING OF DATA MEMORY CHARACTERS FOR A MATCH WITH "X," "Y," AND "Z" CONTIGUOUS
; CHARACTERS
; 16 4-BIT CHARACTERS ASSUMED STORED IN M(0,15) THROUGH M(0,0)
; "X," "Y," AND "Z" CHARACTERS ASSUMED STORED IN AND ASSIGNED VALUES OF M(1,15) THROUGH M(1,13), RESPECTIVELY
; NORMAL RETURN IF NO MATCH
; RETURN THEN SKIP IF MATCH OCCURS WITH THE ACCUMULATOR CONTAINING THE DIGIT NUMBER OF "X"

```

```

    X = 1,15
    Y = 1,14
    Z = 1,13

SEARCH:
    LBI             0,15 ; POINT TO M(0,15)
LOOKX:
    LDD             X   ; X TO A
    SKE             ; X FOUND?
    JP             NOX  ; NO, JUMP TO X
    XDS             ; YES, POINT TO NEXT LOWER DIGIT
    JP             LOOKY ; LOOK FOR Y MATCH, IF AT M(0,0) SKIP AND NORMAL RETURN — NO MATCH
NOX:
    LD              ; DECREMENT DIGIT POINTER
    XDS
    JP             LOOKX ; LOOK AGAIN FOR X MATCH, IF AT M(0,0), SKIP AND NORMAL RETURN — NO
    RET             ; MATCH
LOOKY:
    LDD             Y   ; Y TO A
    SKE             ; Y FOUND?
    JP             LOOKX ; NO, TRY AGAIN
    XDS             ; YES, POINT TO NEXT LOWER DIGIT
    JP             LOOKX ; LOOK FOR Z MATCH, IF AT M(0,0), SKIP AND NORMAL RETURN — NO MATCH
    RET
LOOKZ:
    LDD             Z   ; Z TO A
    SKE             ; Z FOUND?
    JP             LOOKX ; NO, TRY AGAIN
    OBA             ; YES, MATCH COMPLETE, COPY Z DIGIT ADDRESS TO A
    AISC             2  ; ADD 2 TO A TO EQUAL X DIGIT ADDRESS
    RETSK          ; RETURN THEN SKIP — MATCH FOUND

```

## 4.11 Programming Techniques for the COP421-Series, COP410L and COP411L

### COP421-Series Programming

Since the COP421-series differs from the COP420-series only in not having the  $IN_3$ - $IN_0$  inputs, the foregoing programming considerations and examples for the COP420-series are, for the most part, relevant to COP421-series programming. However, due to its lack of IN inputs, the COP421-series does not include the ININ instruction, and its INIL instruction inputs only CKO into A (when CKO is programmed as a general-purpose input). The following are the results of these COP421 differences:

1. MICROBUS™ interface programming is not available since  $IN_3$ - $IN_0$  cannot be mask-programmed as WR, CS, and RD, respectively. Also,  $G_0$  cannot be mask-programmed as a "ready" output to facilitate "handshaking" with a host CPU over the MICROBUS™ bus. The COP421 may still, however, function as a CPU peripheral component, relying on more general, programmed I/O techniques.
2. Due to the lack of IN inputs, other bidirectional I/O pins must be used as general purpose input pins when implementing a programmed input operation.
3. A hardware interrupt utilizing  $IN_1$  is not possible. (Setting  $EN_1$  has no effect on the operation of any COP421.) Any interrupt servicing must be accomplished using software interrupt techniques. (The routine provided in Section 4.8 is inapplicable to the COP421-series.)
4. A software interrupt cannot rely on the inputting and testing of the  $IL_3$  or  $IL_0$  latches associated with  $IN_3$  and  $IN_0$  inputs. Software interrupts, therefore, require that the interrupt signal be tied to one of the non-latched input pins. As a result, the input interrupt signal must be input and tested at least once during each "low" and "high" pulse occurring during each period of the signal. For example, if the interrupt signal is a 50% duty cycle, 60 Hz square wave, it must be tested at least twice every  $\frac{1}{60}$  second.

### COP410L/COP411L Programming

Since the COP410L/COP411L, as with the COP421-series, does not have IN inputs, the above programming considerations relating to the COP421 apply as well as to COP410L/COP411L programming. Also, since, as discussed below, other hardware logic elements are not included in the architecture of the COP410L, the following additional considerations apply to COP410L programming:

1. The COP410L/COP411L has one-half the ROM and RAM of the COP420-series and COP421-series. ROM, therefore, consists of  $512 \times 8$ -bit

words, limiting program code to eight pages (pages 0-7). RAM consists of a  $32 \times 4$ -bit RAM, organized as four RAM registers (0-3) consisting of 8 4-bit digits (9-15,0). The LBI register reference instruction should, therefore, contain a "d" field equal to 9-15 or 0. Since all LBIs will reference RAM digits 9-15 or 0, all LBIs are single-byte instructions, occupying one word in program memory. A field restriction occurs with respect to the memory reference XAD instruction: only an XAD 3,15 instruction is valid, limiting its use to reference a RAM "scratch-pad" digit *contained in M(3,15) only*.

2. The COP410L/COP411L has 2 subroutine save registers, SA and SB. Only two levels of subroutine nesting, therefore, are allowed. The programmer should also realize that since LQID pushes and pops the stack in performing the operation associated with this instruction, only 1 level of subroutine nesting should be in effect at the time of the execution of this instruction. (Otherwise the second level of previous subroutine nesting will be disrupted — the previous contents of SB will be lost.)
3. Since the COP410L/COP411L does not have an internal divide-by-1024 time-base counter, the SKT instruction is not available. "Real-time" routines, such as 12-hour timekeeping and the like, must rely on external time-base inputs in order to derive a time-base for such routines (e.g., external 50/60 Hz input for time-of-day routines).
4. Certain deleted or altered instructions have already been mentioned: INIL, ININ, and SKT are not available; LBIs must have a "d" field equal to 9-15 or 0, and XAD's operand must equal 3,15. The following instructions have also been deleted from the COP410L/COP411L instruction set. To the right of each of the following deleted instructions, where appropriate, alternative COP410L/COP411L instructions are shown which, when executed in succession, will perform the same or similar operation as the deleted instruction:

Deleted Instructions	Alternative COP410L/COP411L Instructions
LDD	LBI, LD
CASC	COMP, ASC
ADT	AISC 10, NOP
CQMA	INL
OGI	OMG
XABR	
SKT	
ININ	
INIL	

For further information on deleted or altered COP410L/COP411L instructions and the operations performed by the alternative instructions given above, see Section 3.4.

# 5 COP400 I/O Techniques



This chapter provides information and examples pertaining to hardware and software interfacing techniques for the COP400 Microcontrollers. The information contained in this chapter is derived, in large part, from material already provided in previous chapters, particularly Chapter 2. The reader should refer to this chapter when reading the following material to obtain a complete picture of the COP400 series I/O characteristics and capability.

The following text provides I/O examples for the COP420 specifically. The I/O capability of the other members of the COP420-series (e.g., COP420L and COP420C), the COP444L and other, less inclusive devices, the COP410L and COP411L, are summarized in Table 5.1.

## 5.1 Hardware Interfacing Techniques

### COP420 I/O

Figure 5.1 depicts the I/O lines associated with the COP420. As indicated, there are 24 I/O lines. The following discussion provides information on the capabilities of the mask-programmable I/O options associated with the COP420. These optional configurations are shown in Figure 5.2.

### COP420 Inputs

COP420 inputs may be programmed either with a depletion-load device to  $V_{CC}$  or floating (Hi-Z input). All inputs are TTL/CMOS compatible. Hi-Z inputs should not be left floating; they should be connected to the output of a "high" and "low" driving device if active or to  $V_{CC}$  or ground if unused. Inputs may also be optionally programmed for higher trip levels for interfacing to non-TTL sources (e.g., keyboards, switches).

Table 5.1 COP400 Comparison Chart

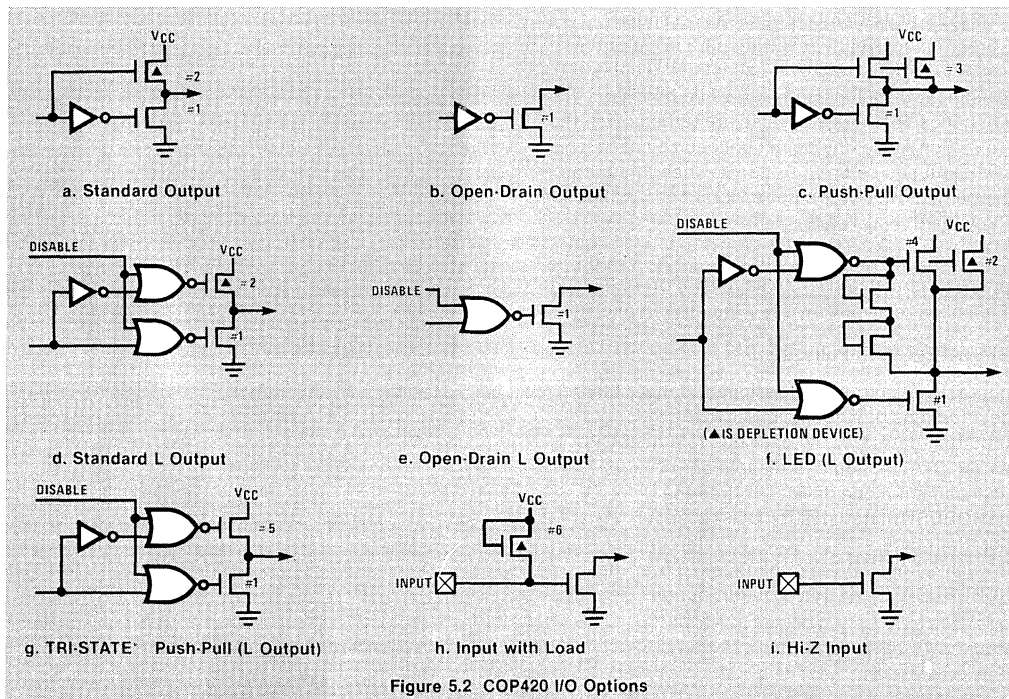
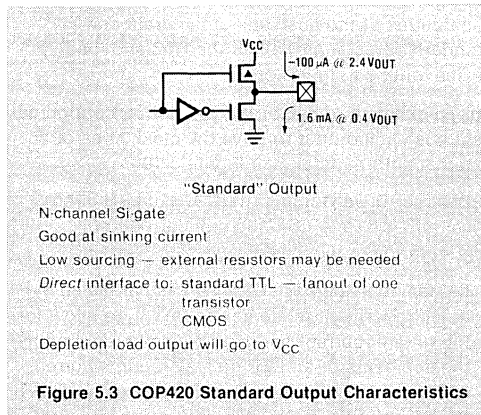
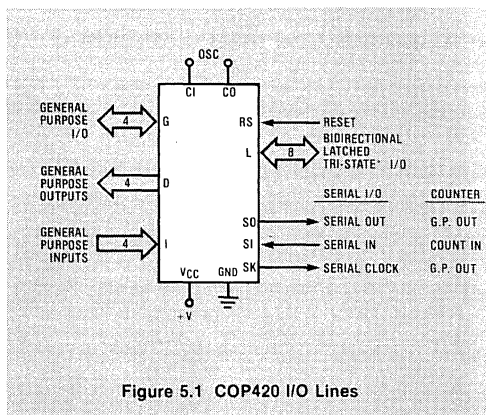
I/O Pins	Bits	COP420	COP420C	COP420L	COP410L
D <sub>OUT</sub>	4	TTL	LSTTL	20mA Sink	20mA Sink
G <sub>OUT</sub>	4	TTL	LSTTL	20mA Sink	LS TTL
L <sub>OUT</sub>	8	TTL or LED	LSTTL or LED	LS or LED	LS or LED
SO, SK	2	TTL	LSTTL	LS	LS
IN	1	4 Inputs	4 Inputs	4 Inputs	No
SI	1	Shift Register or Counter Input	Shift Register	Shift Register or Counter Input	Shift Register or Counter Input
CKI	1	Oscillator Input	Oscillator Input	Oscillator Input	Oscillator Input
CKO	1	Oscillator Out or SYNC In or General In or RAM Supply	Oscillator Out or General In	Oscillator Out or SYNC In or General In or RAM Supply	Oscillator Out or SYNC In or RAM Supply
RESET	1	RESET Input	RESET Input	RESET Input	RESET Input
V <sub>CC</sub> , GND	2	Power Supply	Power Supply	Power Supply	Power Supply
Oscillator Frequency Range		0.4 to 4MHz	32kHz to 2MHz	0.2 to 2MHz	200 to 500kHz
Cycle Time		4 to 10 $\mu$ s	15 to 250 $\mu$ s	15 to 40 $\mu$ s	15 to 40 $\mu$ s
V <sub>CC</sub> Supply		4.5 to 6.3V	2.4 to 6.3V	4.5 to 9.5V	4.5 to 9.5V
V <sub>CC</sub> Current (max)		25mA	1mA (25 $\mu$ A)	8mA	5mA

### COP420 Outputs

**Standard Output:** The N-channel device to ground is good at sinking current and is compatible with the sinking requirements of 1 TTL load (1.6 mA at 0.4V); it will meet the "low" voltage requirements of CMOS logic. All output options use this device (device #1), as illustrated in Figure 5.2, for current sinking. The depletion-load device to  $V_{CC}$  provides low sourcing capability (100  $\mu$ A at 2.4V). While this device meets the sourcing requirements of TTL logic and will go to  $V_{CC}$  to meet the "high" voltage requirements of CMOS logic, an external resistor to  $V_{CC}$  may be required to interface to other external devices requiring higher sourcing capability. A standard output may be connected directly to the

base of an external transistor for current sourcing since the depletion-load device's current capability is limited to a safe operating area. Figure 5.3 provides a summary of the characteristics of the COP420 Standard Output.

**Open-Drain Output:** The COP420 open-drain output uses the same enhancement mode device to ground as the standard output with the same current sinking capability. As its name implies, this output configuration does not contain a load device to  $V_{CC}$ , allowing various external pullup techniques as required by the user's application.



**Push-Pull Output:** The COP420 push-pull output differs from the standard output configuration in having an enhancement mode device in parallel with the depletion-load device to  $V_{CC}$ , providing greater current sourcing capability and faster rise and fall times when driving capacitive loads. This option is available for the COP420 SO and SK outputs, often tied to the highly capacitive clock lines of external shift registers to provide additional external I/O for the COP420. (For an example, see Figure 5.20.) If a push-pull output is interfaced to an external transistor, a limiting resistor must be placed in series with the base of the transistor to avoid excessive source current flow out of the push-pull output.

Figure 5.4 summarizes, in interconnect form, the information provided above relevant to the capabilities of the push-pull, open drain and standard outputs, as well as the Hi-Z and load device input configurations.

For an example of use of the SK output, configured as a push-pull output to drive the clock lines of an external shift register, see Figure 5.10.

**LED Direct Drive Output:** The COP420 LED direct drive output differs from the standard output configuration in two basic ways:

1. Its depletion-load device to  $V_{CC}$  is paralleled by an enhancement mode device to  $V_{CC}$  to allow for the greater current sourcing capacity required by the segments of an LED display. Source current is clamped to prevent excessive source current flow.
2. This configuration can be disabled under program control by resetting bit 2 ( $EN_2$ ) of the enable register to provide simplified display segment blanking. However, while both enhancement mode devices are turned off in the disabled mode, the depletion-load device to  $V_{CC}$  will still source up to 0.125mA when this output is turned off. (This is not a worst case pull-up for keyboard input loads).

For an example of use of the L I/O ports, using this option, to directly drive the segments of a LED and VF display, respectively, see Figures 5.11 and 5.12.

**TRI-STATE® Push-Pull Output**

This COP420 output was designed to meet the specifications of National's MICROBUS™, outputting data over the data bus to a host CPU. It has TRI-STATE® logic to disable both enhancement mode devices to free the MICROBUS™ data lines for COP420 input operation. Figure 5.13 shows an interconnect between a host CPU and the COP420 over the MICROBUS™ using this L output option.

**COP420 I/O Summary**

Figures 5.5 through 5.9 provide diagrams of the internal logic and a summary of the hardware and software features associated with the COP420 I/O ports.

**Interconnect Examples**

Figures 5.10 through 5.14 provide interconnect diagrams illustrating several schemes for interconnecting the COP420 to external devices. Several of these interconnect diagrams, with minor variations, are used in providing software I/O techniques in the final sections of this chapter.

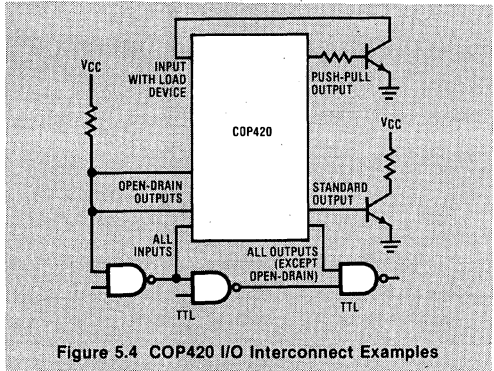
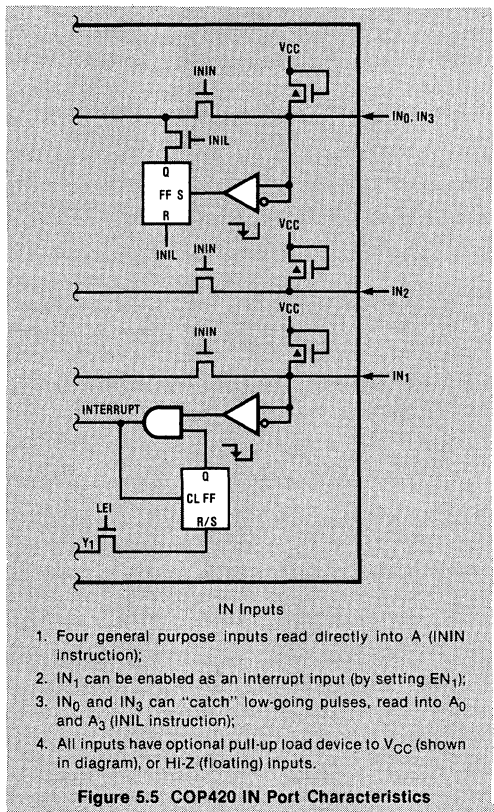
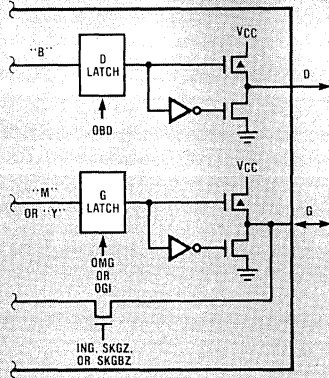


Figure 5.4 COP420 I/O Interconnect Examples



1. Four general purpose inputs read directly into A (ININ instruction);
2.  $IN_1$  can be enabled as an interrupt input (by setting  $EN_1$ );
3.  $IN_0$  and  $IN_3$  can "catch" low-going pulses, read into  $A_0$  and  $A_3$  (INIL instruction);
4. All inputs have optional pull-up load device to  $V_{CC}$  (shown in diagram), or Hi-Z (floating) inputs.

Figure 5.5 COP420 IN Port Characteristics



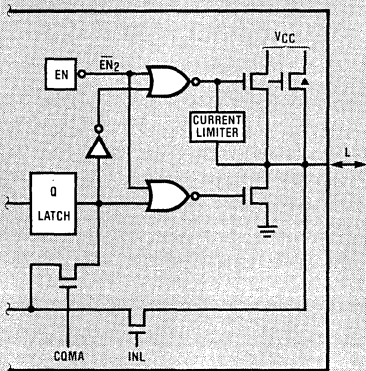
D Outputs

1. Four general purpose outputs loaded from B (OBD instruction);
2. Standard (as shown) or open-drain outputs.

G Inputs

1. Four general purpose I/O lines loaded from memory (M) by OMG instruction or loaded with immediate data (Y) by OGI instruction;
2. Read inputs into accumulator (ING instruction), test individually (SKGBZ instruction), collectively (SKGZ instruction) for zero — seg G latch to "1" when using as input;
3. Standard (as shown) or open-drain outputs.

Figure 5.6 COP420 D and G Port Characteristics

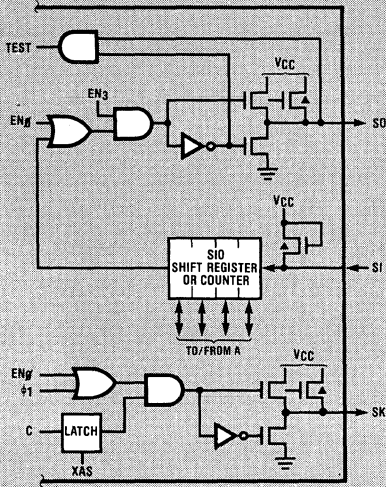


L TRI-STATE\* Inputs/Outputs

1. Eight TRI-STATE inputs/outputs, loaded with Q latch data by setting EN<sub>2</sub> or direct input of L port data to M and A (INL instruction); Q latch loaded from A and M by CAMQ instruction and read into M and A by CQMA instruction;
2. L ports TRI-STATED with EN<sub>2</sub> = 0 (if output contains depletion-load device to V<sub>CC</sub>, I<sub>OL</sub> ≈ 0.2mA @ 0V in);
3. All output options available:
  - a. Standard
  - b. Open-Drain
  - c. Push-Pull
  - d. LED Direct Drive (as shown)
  - e. TRI-STATE Push-Pull

Figure 5.7 COP420 L I/O Port Characteristics

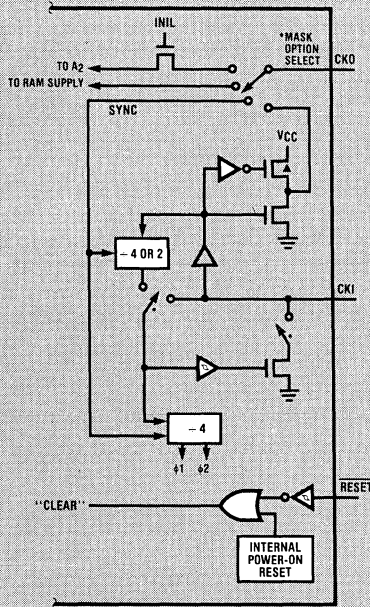




SI Input, SO, SK Outputs

1. SI is a single-pin input to the SIO register. SIO can be enabled as a 4-bit serial shift register or a 4-bit binary counter, selected by EN<sub>3</sub>.
2. If SIO is selected as a counter, SO outputs the value of EN<sub>3</sub>; SK outputs the value of C upon the execution of an XAS instruction.
3. If SIO is selected as a shift register, SO may be used as a serial data output and SK may be a logic controlled clock selected by EN<sub>3</sub>.
4. The contents of SIO may be exchanged with A using an XAS instruction.
5. SI, SO and SK are also used for "In-house" standardized testing of the COP420.
6. SI may be configured with a load device to V<sub>CC</sub> (as shown) or as a Hi-Z input.
7. SO and SK may be configured as:
  - a. Standard
  - b. Open-Drain, or
  - c. Push-Pull (as shown) outputs.

Figure 5.8 SI, SO, SK Characteristics



CKO, CKI and RESET Pins

1. The COP420 CKO pin has the following options:
  - a. output to crystal oscillator;
  - b. general purpose input (read into A<sub>2</sub> by an INIL instruction);
  - c. synchronization (SYNC) input;
  - d. RAM power supply pin.
2. CKI has the following options:
  - a. crystal oscillator input;
  - b. external oscillator input;
  - c. RC controlled oscillator input.
3. RESET may be used as an external reset pin or, if the power supply rise time is greater than 1 ma, as a power-on clear input.

Figure 5.9 COP420 CKO, CKI, RESET Characteristics

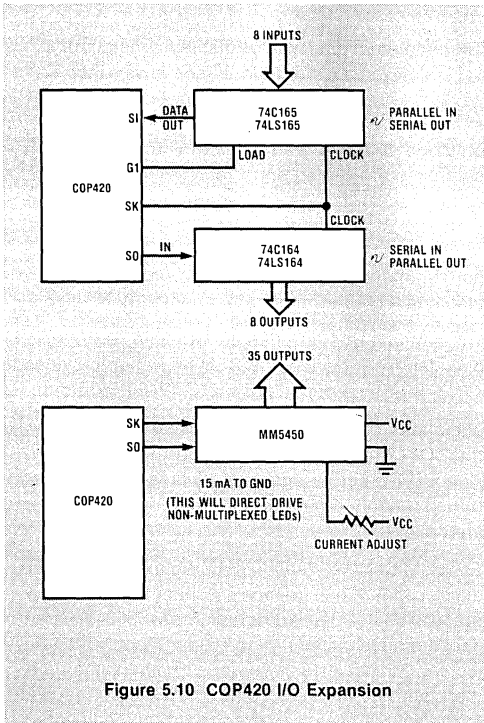


Figure 5.10 COP420 I/O Expansion

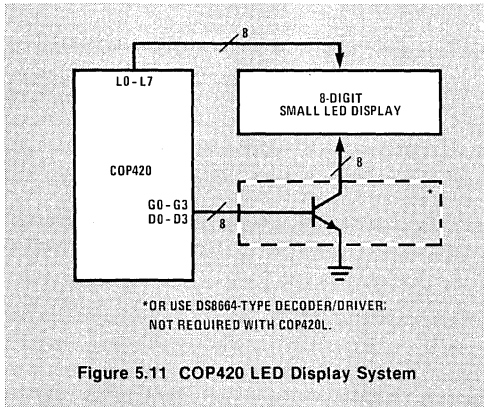


Figure 5.11 COP420 LED Display System

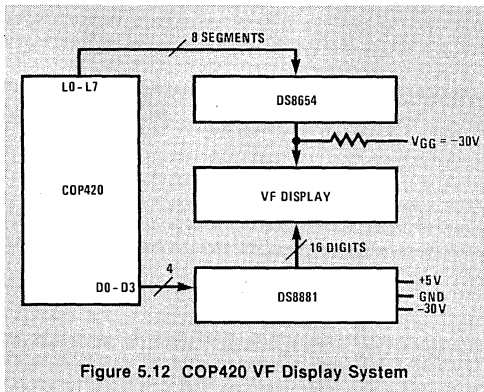


Figure 5.12 COP420 VF Display System

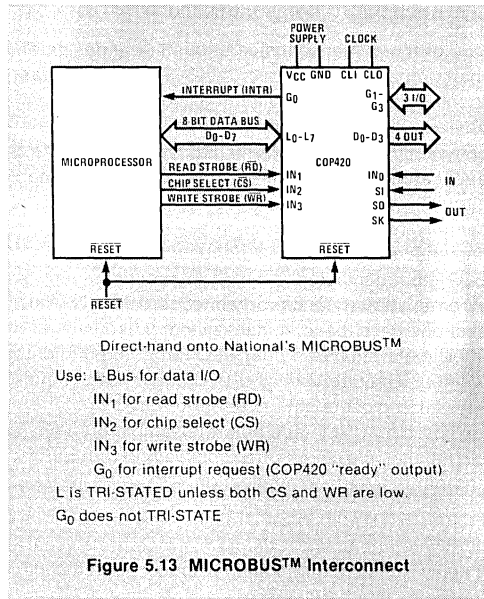


Figure 5.13 MICROBUS™ Interconnect

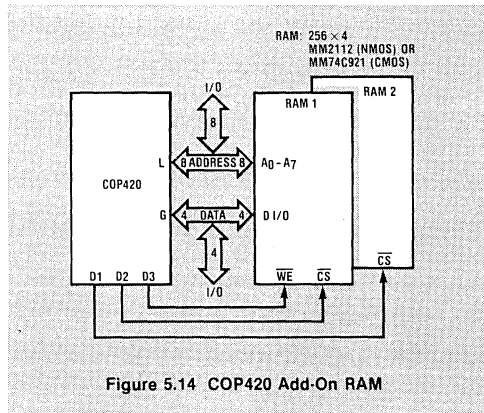


Figure 5.14 COP420 Add-On RAM

### COP400 I/O Comparison Table

Table 5.1 provides a comparison table of the I/O capabilities of COP400 series devices. It should be understood that this is a partial listing of COP400 devices, since more inclusive parts (the COP440 and its related devices) as well as other devices will be available in the near future. For complete information on the listed devices, as well as other members of the COP400 Microcontroller family, consult the appropriate data sheets.

### 5.2 Software I/O Techniques

The following sections of this chapter provide several software I/O examples and techniques for interfacing the COP420 to external I/O, including program code necessary to service these peripherals.



### 5.3 Keyboard/Display Interface

One of the primary considerations in the design of the internal architecture of the COP400 family was to allow for easy interface to keyboards and numeric displays, the input and output peripherals commonly associated with small system applications, using a minimum amount of external circuitry. To further aid in the implementation of such systems, the instruction set was carefully designed to service these peripherals and handle BCD data manipulation with a minimum amount of external circuitry and program code. The following sections describe a typical keyboard/display interface system to output BCD data stored in data memory (RAM) to a 14-digit LED display, and input keyswitch closure data entered from a 4 × 4 keyboard matrix. In addition, the sample program also makes provision for a timekeeping routine, another typical user application.

Figures 5.15 through 5.18, respectively, provide the hardware interconnect diagram, program flowchart, display timing diagram and assembly source code for the basic interface scheme. The general approach of the interface is common to most keyboard/display interfaces. It takes advantage of the fact that an image persists in the eye for a fraction of a second after the source is removed. It is not necessary, therefore, to have all display digits on simultaneously: the digits are sequentially enabled (multiplexed) at a rate fast enough to avoid noticeable flicker. Multiplexing greatly reduces the amount of interconnect and buffer hardware required.

The most common type of display consists of several seven-segment digits (see lower right section of Figure 5.15). Each light emitting diode segment has two terminals and conducts current in only one direction. Various combinations of segments are turned on to represent numbers and a few alphabetical characters. In our example, the cathodes of all segments (Sa-Sg, D.P.) in a given digit are connected together and the anodes of corresponding segments of the different digits are also connected together (common cathode display).

The cathode or digit lines are driven by a decoder/driver device, the DS8664, which provides a 4-to-14 buffered decode of the COP420 D outputs.

The anode or segment lines are driven directly by the COP420 L I/O ports, utilizing the L output LED Direct Drive output option. A given segment is turned on only if both its digit and segment lines are driven.

Each digit of the display is multiplexed, with each digit scanned in sequence by changing the binary output code at the D outputs. The DS8664 decoder/driver will set a corresponding D line to a low level to drive each cathode. At the same time the L outputs are set at a high level to correspond to the values necessary to turn on the segments associated with the numeric or alphabetical character to be displayed for the present digit. (To display a "3" at digit 5, segments Sa, Sb, Sc, Sd and Sg would be driven high when D<sub>5</sub> is driven low.)

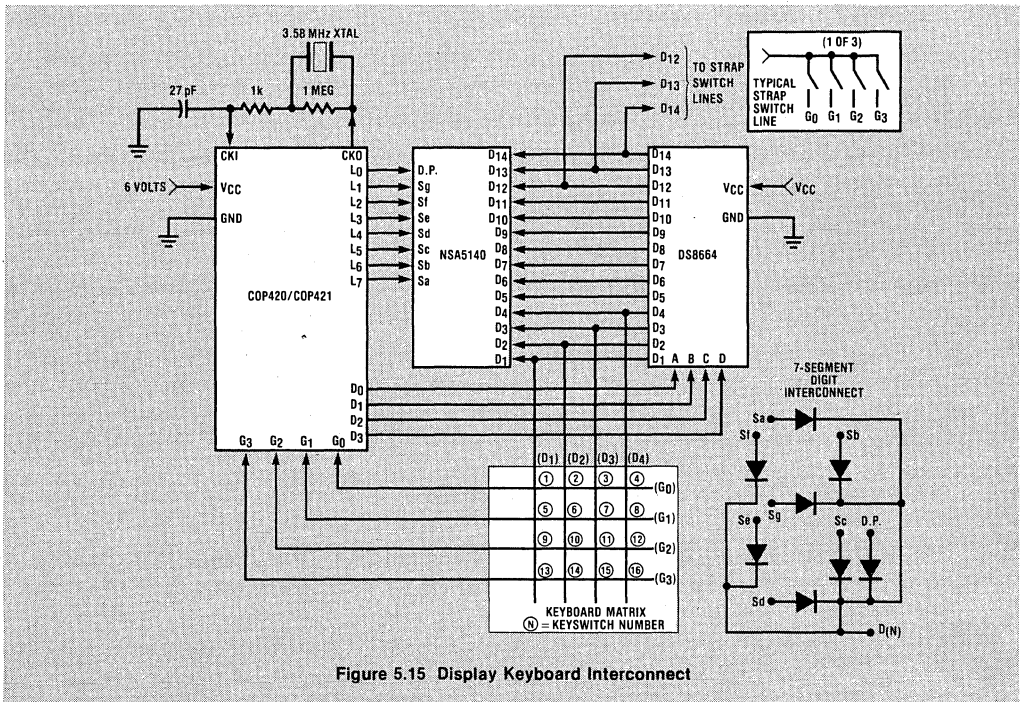


Figure 5.15 Display Keyboard Interconnect



The following is a list of design criteria and considerations relevant to the sample keyboard/display interface:

1. With this design, if two keys on different G I/O lines are pressed simultaneously, key identity may be lost. After sensing a key closure, the program requires that the keyboard be clear (no keys pressed) for a short duration before it will input another key. "Rollover" and "shift-key" schemes may be implemented with more sophisticated designs.
2. Multiple key closures on the same G I/O line will allow segment current to flow through the keyboard causing display digits to be ANDed. Key closure is still detected, however, because the "on" driver presents a small resistance to GND compared to the resistance that the "off" driver and G port present to  $V_{CC}$ . The ANDing of display digits may be prevented by placing diodes on each digit line. If key identity must be maintained when more than two keys are closed, a diode must be placed in series with each keyswitch.
3. For this design, the G ports are configured as standard outputs (options 21 - 24 = 0). The program itself sets them each to "1" at the beginning and on each pass through the main program loop. When all keys in the associated matrix row are up, the port will read as a "1." When a key is closed, its corresponding D line will pull the associated G port low, with a "0," therefore signifying key closure.
4. The L ports are configured as *LED Direct Drive* outputs (options 5 - 8 and 12 - 15 = 2) to directly drive the segments of the LED display. An average L output source current capability of 8mA is assumed, being midway between the minimum (2.5mA) and maximum (14mA) current sourcing specifications for this output configuration at  $V_{CC} = 6V$ .
5. To prevent flickering of the display, the display should be refreshed at a rate of at least 100 Hz (1/P in Figure 5.17).
6. The duty cycle (S/P in Figure 5.17) must be maintained to ensure adequate brightness. The L port segment current capability is assumed, as mentioned above, to be 8mA and the NSA5140 requires 0.5mA *average* current. Average current is determined by the segment duty cycle and should be the average display current requirement divided by the peak output current or  $0.5 - 8 = 1/16$ . Therefore, the program must be written to ensure a duty cycle of at least 1/16 for proper LED display brightness.
7. Each segment on time (S in Figure 5.17) must be the same width to ensure that all digits are *uniformly* bright.
8. Since keyswitches bounce, the program must debounce or filter the signals on the G lines. This is achieved by requiring that a key be held down for at least four display cycles before being accepted. A key must also be lifted for at least four display cycles before a new key can be accepted.
9. To prevent crosstalk or ghosting between display digits, a LED display requires segment blanking (Sb in Figure 5.17).

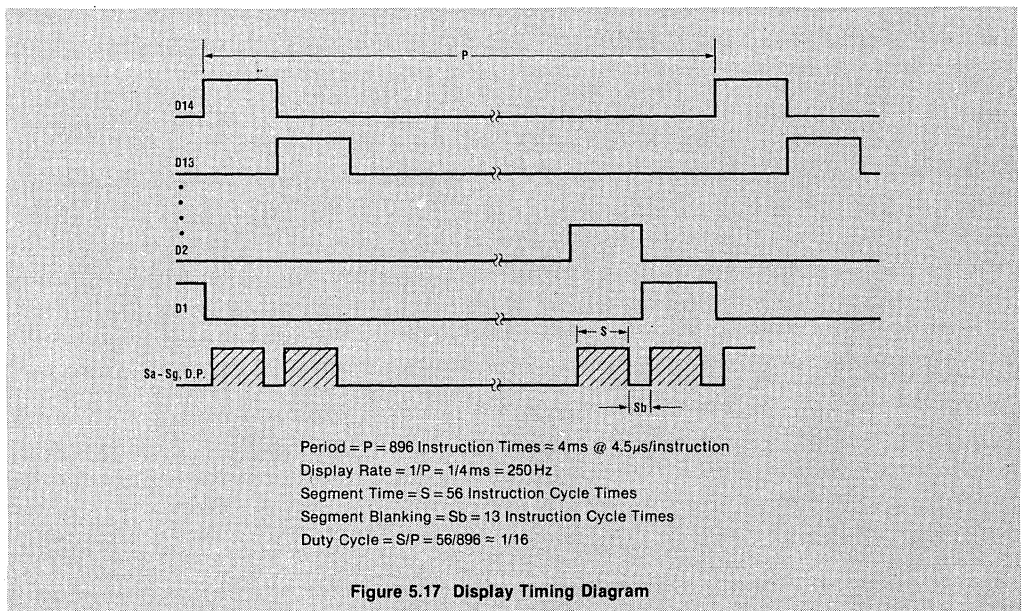


Figure 5.17 Display Timing Diagram

10. The system clock oscillator is configured as a crystal controlled oscillator with the instruction cycle frequency derived by driving the crystal oscillator frequency by 16 (options 2 and 3 = 0). This interface scheme uses an inexpensive 3.58MHz TV crystal to provide the clock oscillator frequency, divided by 16 to derive a 4.5 $\mu$ s instruction cycle time. This also allows use of the "TIMEKP" (timekeeping) routine given in Section 4.9, which uses the internal COP420 Time-Base Counter and the SKT instruction, together with program compensation techniques, to provide a "real time base" for keeping time — eliminating the need for an external 60Hz real-time input and associated external circuitry.

### Sample Display/Keyboard Debounce-Decode Program

Figure 5.16 depicts the flowchart for the sample display/keyboard debounce routine. The actual assembly source code written to perform the flowchart operations is given in Figure 5.18.

Following the flowchart from top to bottom, and referring to the source code where appropriate, the following sequence of operations is performed:

1. The G port is set to 15 (each G line set to "1"). This allows them to be driven low when scanned by their associated D lines. If a keyswitch is closed, the associated G line will therefore become a "0," to be input and tested by the keyboard servicing routine.
2. The program initializes the KBC (Keyboard Debounce Counter) to 15 (1111<sub>2</sub>). This counter name, as well as two other RAM status digit names, "DIGIT" and "STORE," are assigned the values of their RAM register and digit numbers by assembler assignment statements at the beginning of the source code. This allows these names to be substituted in the operand field of instructions which reference these RAM digits, providing more effective documentation of the source code program. For example, since the KBC is located in RAM register 3, digit 15 and since this value (3,15) must be contained in the operand field of an instruction referencing the KBC, an assignment statement of KBC = 3,15 is written at the beginning of the program. Thereafter, an instruction referencing the KBC may use its name, rather than its RAM value, in the operand field of the instruction (e.g., an LBI KBC will be interpreted by the assembler as an LBI 3,15).

The contents of the KBC are depicted in the upper right hand corner of the flowchart. From left to right, the bits of the counter indicate the following status conditions: the "up" bit, set to "1" if all keys are up; the "not ready" bit (NRB), set to "1" if keyswitch data has not been debounced; two binary counter bits, both set to "1" at the beginning of the debounce sequence.

As will be seen, the two leftmost bits of the KBC ("up" and "NRB") are tested during the debounce routine to determine which branch of the routine will be executed. The rightmost bits, the binary counter bits, provide a binary count of the number of times the program falls through the debounce routine.

3. The internal time-base counter is tested for overflow by an SKT instruction, calling the "TIMEKP" subroutine given in Section 4.9 to keep time if the SKT instruction tests "true."
  4. The digit position is set to 14, the most significant digit of the display. As indicated by the source code, the digit position is set by loading Bd (the RAM digit-select register) with the digit position value with an LBI instruction. Bd is later output to the D ports using an OBD instruction, decoded by the DS8664 to enable the appropriate display digit line (D<sub>14</sub>-D<sub>1</sub>). Since, as mentioned, Bd also functions as a pointer to a particular RAM digit as well as being the source of a direct output of data to the D ports, loading Bd also is used to access the contents of a particular RAM digit, used later by an LQID instruction to obtain seven-segment decode data contained in a lookup table. Because of this dual function of Bd, the segment data for a particular *display* digit should be located in a numerically corresponding *RAM* digit of the RAM display register (register 0). For example, when Bd is set to 14 by an LBI 0,14 to later enable *display* digit 14, it will also be used to obtain the segment lookup data for that *display* digit located in *RAM* register 0, digit 14. Consequently the segment data pointers for *display* digits 14 through 1 are located in RAM register 0, digits 14 through 1, respectively.
- As will be seen below, the segment data contained in a particular RAM digit, although used by LQID to obtain the actual seven-segment data output to the display, will equal the binary equivalent of the numeral to be displayed (e.g., if a RAM register 0 digit contents = 0010<sub>2</sub>, the LQID instruction will access the seven-segment diode data for the numeral "2." RAM digit contents equal to 10-15 will be used to access special seven-segment alphabetical characters.
5. The value of the digit position loaded into Bd is saved in M(1,15), equated by an assembler statement, as explained in 1. above, to the symbol name "DIGIT." The digit value is saved for later manipulation by the display program (testing, decrementing).
  6. The segments of the display are blanked, a requirement for LED multiplexed displays. This is accomplished by disabling the drivers from the Q latches (which contain the seven-segment decode display data) to the L ports by resetting bit 2 of the EN-register with an LEI 0 instruction.

With the L drivers thus disabled, the L I/O ports are disabled, turning off the segments of the display.

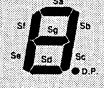
- Next, the program utilizes an LQID instruction to access and load seven-segment decode data contained in a lookup table into the Q latches. This is accomplished in the following manner: as explained in Section 3.2, LQID loads Q<sub>7</sub>-Q<sub>0</sub> with the 8-bit contents of ROM (I<sub>7</sub>-I<sub>0</sub>) pointed to by P<sub>9</sub>, P<sub>8</sub>, A and M. In this example, LQID is located in page 0, with the result that, at the time of execution, P<sub>9</sub>, P<sub>8</sub> = 0,0. The program sets A = 0100 with an AISC 4 instruction before execution of the LQID instruction so that P<sub>7</sub>, P<sub>6</sub> = 0,1 and P<sub>5</sub>, P<sub>4</sub> = 0,0. Since the upper 4 bits of P may be thought of a ROM "page-select" bits, selecting 1 of 16 pages (0-15) and, since these 4 bits will equal 0001 at the time of the execution of the LQID instruction, it will always "look to" page 1. The lowest 6 bits of P (P<sub>5</sub>-P<sub>0</sub>) may be thought of a ROM "word-select" bits, selecting 1 of 64 (0-63) words on a "looked-to" page. Moreover, P<sub>5</sub> and P<sub>4</sub>, the upper 2 bits of these 6 word-select bits, may be thought of as ROM "sub-page-select" bits, selecting 1 of 4 (0-3) successive groups of 16 words on a 64-word ROM page. Since P<sub>5</sub> and P<sub>4</sub> will always equal 0,0 upon the execution of the LQID instruction, it will always look to one of the first 16 words located in page 1. Since the contents of M (the RAM digit pointed to by the B register), are loaded into the lowest 4 bits of P (P<sub>3</sub>-P<sub>0</sub>), it is the binary contents of M directly (0-15) which determine which of the first 16 words (0-15) on page 1 are "looked up" and placed in Q.

In effect, M is the only variable involved in the LQID operation with its contents directly determining which one of the 16 words in page 1 (words 0-15) are loaded into Q. Of course, the seven-segment decode values have been placed in these locations. Also, as indicated above, the first 10 words (locations 0-9) have been loaded with the seven-segment decode values for the numerals 0-9, respectively. Consequently if M = 3 binary (0011<sub>2</sub>), a LQID will place the seven-segment lookup data for a display numeral 3 into Q. If M = 10-15 binary, LQID will place the seven-segment decode values for the special alphabetical characters P, A, U, C, F and E, respectively, into Q, since page 1, locations 10-15, contain the decode values to display these characters on the display.

The hexadecimal value of the seven-segment lookup data is placed in page 1, locations 0-15 with the assembler .WORD directive. Although operands of the .WORD may be concatenated (i.e., .WORD X'FD, X'1F, ...), each 8-bit segment decode value has been placed in successive memory locations with a separate .WORD directive. It should be noted, as indicated by the

comments to the program, that ROM word bits I<sub>7</sub>-I<sub>0</sub> (rightmost to leftmost) represent and are tied via the L ports to the Sa-Sg, D.P. segments of the display. A "1" bit for a particular segment means that that segment will be turned on. In all cases, each seven-segment decode word has the D.P. bit (I<sub>0</sub>) seg; if not later reset by the program the decimal point segment of a particular digit will be turned on when that digit is serviced. See Table 5.2 for a representation of the interconnection of the seven-segments of a display digit and a list of binary and hex values associated with setting the segments of a digit to display the numerals 0-9.

Table 5.2 Seven-Segment Decode Values



Display	Binary Values							Hex Values	
	Sa	Sb	Sc	Sd	Se	Sf	Sg	Sa-Sdp →I <sub>7</sub> -I <sub>0</sub>	Sdp-Sa →I <sub>7</sub> -I <sub>0</sub>
0	1	1	1	1	1	1	0	0	FC 3F
0.	1	1	1	1	1	1	0	1	FD BF
1	0	1	1	0	0	0	0	0	60 06
1.	0	1	1	0	0	0	0	1	61 06
2	1	1	0	1	1	0	1	0	DA 5B
2.	1	1	0	1	1	0	1	1	DB 5B
3	1	1	1	1	0	0	1	0	F2 4F
3.	1	1	1	1	0	0	1	1	F3 4F
4	0	1	1	0	0	1	1	0	66 66
4.	0	1	1	0	0	1	1	1	67 66
5	1	0	1	1	0	1	1	0	B6 6D
5.	1	0	1	1	0	1	1	1	B7 6D
6	1	0	1	1	1	1	1	0	BE 7D
6.	1	0	1	1	1	1	1	1	BF 7D
7	1	1	1	0	0	0	0	0	E0 07
7.	1	1	1	0	0	0	0	1	E1 07
8	1	1	1	1	1	1	1	0	FE 7F
8.	1	1	1	1	1	1	1	1	FF 7F
9	1	1	1	0	0	1	1	0	E6 67
9.	1	1	1	0	0	1	1	1	E7 67

- A comparison is made to see whether the decimal point position stored in RAM is equal to the digit position of the digit to be displayed during the present pass through the display loop. If the comparison result is "false," the program jumps to "NODP," which resets the least significant bit of Q to keep the decimal point segment of the current digit off when Q latch data is later output to the display via the L ports. Note that an X instruction must follow the CQMA and precede the CAMQ instruction to maintain the integrity (bit-weights) of the Q data, since these instructions perform opposite exchanges with respect to A and M. (See Section 3.2.)
- If the comparison tests "true," the least significant bit of Q is left set to turn on the decimal point of the current digit and a delay is added to ensure that the program will require

the same amount of execution time whether or not the comparison tests "false" (goes to "NODP") or "true." This and other delays contained in the program ensure that the servicing of a particular display digit will always require the same number of instruction cycle times regardless of which branch of the program is executed during a pass through the program; this is necessary for equal segment-on time for each digit and uniform brightness among the various digits of the display.

10. Digit position data is output from Bd to the D outputs, decoded by the DS8664, enabling the appropriate digit of the display and scanning the corresponding D line (if connected) to the keyboard matrix column or strap switch line.

11. Segment data is output to the current digit by enabling the L drivers with an LEI 4 instruction, setting bit 2 of the EN register and outputting the 8-bit Q latch data to the L I/O ports, the latter connected directly to the segments of the display.

12. Having output data to one digit of the display, the program now begins to service the keyboard. A test is made to see whether any key closure has occurred. If so, the program jumps to "KEYDWN," first testing to see if the key closure occurred on a strap digit line. If this test result is true, the strap data is read into RAM and the program goes to "NRDY." If the key closure was associated with the keyboard matrix, the "up" bit of the KBC is reset and the KBC is tested for all 4 bits equal to 0. If the KBC equals 0, indicating a debounced keyswitch closure, the program blanks the display, inputs the G port (keyswitch row data) into A, and jumps to the keyboard decode routine. If the KBC did not equal 0, the program also goes to "NRDY" (with the KBC "up" bit reset to indicate a key closure).

It should be noted that the "up" bit is not reset if the key closure was a strap data switch. As will be seen, this means the program will not treat this switch closure as a key depression (since the "up" bit remains set) and does not debounce this closure nor jump to decode a strap switch closure. Strap switches are of the on/off type not requiring debouncing as do the *momentary* on/off keyswitches. Also, a strap switch decode routine, in this example, is not necessary. The strap data bits read into RAM may be tested at any time for execution of a routine implementing the "mode" associated with a particular strap switch closure.

13. If the program jumps to "NRDY," a test is made to determine whether the digit position equals 1, indicating that all 14 digits have been displayed. If the last digit has not been displayed, the digit position is decremented by one and the program goes to "DSP2" to service the next digit. If the

last digit has been displayed, the program falls through to "DEBOUN," the keyswitch debouncing portion of the program.

14. Debouncing begins at "DEBOUN" by testing to see whether the up bit has been reset, indicating a keyswitch closure. If not, the program takes the right branch to "ALLUP" and tests the not ready bit (NRB) of the KBC. If NRB is equal to 1, the KBC is decremented, the up bit remains set and the program goes back to "DSP1" to output data to all 14 digits again. If, on the first pass through the program, no key closure has occurred, the KBC will enter the debounce routine equal to 1111, exiting with a decremented value of 1110. Provided all keys remain up, it will take four passes through the right debounce branch before the KBC has been decremented to 1011, thereby resetting the not ready bit. If all keys remain up *after* four passes, the program will continue to fall through the NRB not equal to 1 (right) branch, keeping the KBC at 1011. The foregoing operations ensure that all keys remain up for at least four debounce passes before the not ready bit is reset to 0 (and a key closure will be accepted for *keydown*-debouncing).

15. If, upon entering the debounce routine, the up bit has been reset indicating a key closure, the program will take the left debounce branch. If the not ready bit has been reset to 0, indicating as explained above that all keys have previously remained up for at least four passes, the program will continue to decrement the KBC, exiting by setting the up bit and going back to "DSP2." Assuming that the right debounce branch has previously decremented the KBC to 1011, "DEBOUN" will be entered with the KBC equal to 0011. (A key closure resets the up bit.) If the key remains down for four passes, the left branch will decrement the KBC to 0000 and go back to "DSP1" with the KBC equal to 1000 (up bit reset). On the next pass, with the keyswitch still down, "KBCTST" will reset the up bit, the KBC will equal 0000 and the program will jump to the keyboard decode routine with the value of the current D line stored in RAM and the G port data in A.

If the left branch of the debounce routine is entered without the keys having been up for at least four passes (NRB equal to 1), the program will set the KBC to 1111, continuing to do so until the key is lifted and remains up for four passes through the right branch of the debounce loop. Consequently, the program requires that a key be down, as well as up, for at least four debounce periods before keyboard data will be accepted and decoded. Since it takes 16 milliseconds to execute four program passes, ample time is provided to debounce even the most inexpensive keyboards.



16. Once a keyswitch closure has been debounced, the program exits to "KEYDEC" (keyboard decode routine). Upon entry to "KEYDEC," G port data is in the accumulator and represents the particular *row* of the keyboard matrix upon which a key closure has occurred. Data memory M(1,15) contains the value of the D line and represents the particular keyboard matrix *column* upon which a key closure has occurred. The conjunction of a particular D line value and the state of a particular G port bit, therefore, define one of sixteen key closures. Only two instructions are necessary to jump to the particular decode routine associated with each key closure based upon the contents of A and M(1,15): a COMP and a JID instruction.

The COMP instruction is necessary to invert the contents of A since a particular key closure will result in one bit of G being driven to "0," with the remaining bits of G set to "1."

Complementing A results in a "1" representing a key closure with the value of A equal to 0001, 0010, 0100, or 1000 (binary) if the key closure occurred on the G<sub>0</sub>-G<sub>3</sub> row lines, respectively. D will equal 0001, 0010, 0011, or 0100 (binary) if the key closure occurred on the D<sub>1</sub>-D<sub>4</sub> lines, respectively. The JID instruction can then use A and M without further manipulation to access key routine *pointers*, provided these pointers have been placed in appropriate ROM locations (those which the JID will access based upon the values of A and M associated with each key).

The operation of the JID instruction is similar to that of the LQID instruction in that it accesses a ROM location based upon the current value of P<sub>9</sub>, P<sub>8</sub>, A<sub>3</sub>, A<sub>2</sub>, A<sub>1</sub>, A<sub>0</sub>, M<sub>3</sub>, M<sub>2</sub>, M<sub>1</sub>, M<sub>0</sub>. JID, however, then uses the contents of this ROM location as a pointer and transfers program control to this "pointed-to" address. The exact location of this address (first instruction of each decode routine) need not be of concern to the programmer provided it resides within the same ROM block as the JID instruction (see Section 4.1); in this example within ROM block 2 (pages 4-7).

The location of each JID key decode routine pointer must correspond with the current value of P<sub>9</sub> and P<sub>8</sub>, and with the value of A (G port data) and M (D line data) associated with each particular key closure. Table 5.3 depicts the various address values of P<sub>9</sub>, P<sub>8</sub>, A and M for each keyswitch closure. The programmer must place, *within these address locations*, the lower 8 bits of the address of the first instruction of each keydecode routine, to allow the JID instruction to automatically transfer program control to one of these instructions. This loading of ROM address pointers with the proper 8-bit data is easily accomplished using the assembler assignment statement and the .ADDR directive.

First, the programmer must specify a label for the first instruction of each keyswitch decode routine — in this example labels "KEY1"-"KEY16" are given for the starting address of keyswitch number 1-16 decode routines, respectively. (No decode *servicing code* is given.) As already mentioned, these decode labels and the code for each decode routine must reside within the same ROM block as the JID instruction (ROM block 2, pages 4-7).

Second, at each pointer address for each key closure as indicated in Table 5.3, an .ADDR directive must be used to place the lower 8 bits of the address of the beginning of each keyswitch decode routine within each pointer location. This is easily accomplished by moving the assembler location counter to the appropriate pointer address using an assignment statement which assigns the location counter (".") to the hexadecimal address of the appropriate JID pointer location. In this example, for instance, the "KEY1" pointer should be located at address X'111. The assignment statement, . = X'111, moves the assembler location counter to this address. The assembler will then generate code into successive memory locations starting at this location until the assembler location counter is again moved.

After moving the assembler location counter to the proper JID pointer address, the 8-bit value of the address of each appropriate keyswitch decode label location is loaded into the pointer address by using an .ADDR directive with an operand specifying the *label* associated with the first instruction of each key decode routine. For example, to load the keyswitch number 1 decode routine starting address into its pointer location, an .ADDR KEY1 directive will place the lower 8 bits of the address of the KEY1 label into the ROM pointer location.

As can be seen, once labels have been given to the beginning of each decode routine and the assembler location pointer has been moved to the proper JID pointer location, a simple .ADDR (label) statement for each label will automatically allow the JID instruction to transfer program control to the appropriate decode routine for each keyswitch immediately after exiting from the DISPLAY/KEYBOARD DEBOUNCE routine (after complementing G data as explained above). In this example, the assembler location pointer need only be moved four times, since each group of 4 JID pointers resides in successive memory locations. (See Table 5.3.)

Of course, the gaps which exist between the JID pointer locations on pages 4-6 are available for use by other portions of program code. To aid the user in understanding the operations of the assignment statements and .ADDR directives in

this sample program, an *assembler output listing* of the program is provided in Figure 5.19, indicating in the leftmost columns the line numbers, memory addresses and 8-bit memory contents associated with the use of these assembler control statements.

For convenience, the "KEY1" - "KEY16" labels are placed in successive double-byte memory locations, jumping back to "DSP1." In a "real" program, each of these labels would be

followed, respectively, by the code required to perform the program operations associated with each key closure. Alternatively, they might still be placed in successive double-byte memory locations if they used a JMP instruction to jump to any location within the 1K ROM area to a routine which serviced the appropriate keyswitch. For further information on the use of the PDS assembler, see Chapter 8, *PDS User's Manual*.

```

; COP420 DISPLAY/KEYBOARD DEBOUNCE/DECODE ROUTINE
; DISPLAYS 14 BCD DIGITS CONTAINED IN M(0,14) THROUGH M(0,1), HIGH-ORDER TO LOW-ORDER, RESPECTIVELY
; DECIMAL POINT POSITION VALUE CONTAINED IN M(0,15)
; DIGIT POSITION CONTAINED IN M(1,15)
; TEMPORARY STORAGE OF 4 BITS OF SEGMENT DATA IN M(1,14)
; KEYBOARD DEBOUNCE COUNTER (KBC) CONTAINED IN M(3,15)
; SEVEN-SEGMENT DECODE ROM LOOKUP DATA CONTAINED IN PAGE 4, WORDS 0 - F
; ROUTINE READS STRAP DATA SWITCHES TIED TO DIGIT LINES 12, 13 AND 14 INTO M(1,12) THROUGH M(1,14) RESPECTIVELY
; EXIT TO KEY DECODE ROUTINE AFTER DEBOUNCING KEYSWITCH CLOSURES WITH DIGIT VALUE IN M(1,15) AND G PORT DATA
; IN A

        .PAGE          0
        DIGIT          = 1,15      ; ASSIGN VALUE 1,15 TO "DIGIT"
        STORE         = 1,14      ; ASSIGN VALUE 1,14 TO "STORE"
        KBC           = 3,15      ; ASSIGN VALUE 1,13 TO "KBC"
        CLRA          ; FIRST INSTRUCTION MUST BE A "CLRA"

DSPLY:  OGI           15          ; SET ALL G PORTS HIGH
        LBI           KBC         ; POINT TO M(3,15)
        STII          15          ; 15 TO KBC

DSP1:   SKT           ; TIME-BASE COUNTER OVERFLOW?
        JP            NOCNT       ; NO COUNTER OVERFLOW
        JSR           TIMEKP      ; YES, CALL TIMEKEEPING SUBROUTINE

NOCNT:  LBI           0,14        ; START DISPLAY AT DIGIT 14

DSP2:   CBA           ; DIGIT POSITION TO A
        XAD           DIGIT       ; STORE IN M(1,15)
        CLRA
        AISC          4           ; SET A2 TO FLIP TO PAGE 1 FOR LOOKUP
        LEI           0           ; BLANK SEGMENTS (RESET EN2)
        LOID          ; LOOKUP TABLE SEGMENT DATA TO Q
        LBI           DIGIT       ; POINT TO DIGIT POSITION
        LD            1           ; DIGIT POSITION TO A, POINT TO DECIMAL POINT POSITION DIGIT

        SKE           ; DECIMAL POINT = DIGIT POSITION?
        JMP           NODP        ; NO, RESET DECIMAL POINT BIT IN Q
        CLRA
        AISC          4
        JP            -1         ; DELAY 9 INSTR. CYCLE TIMES

DIGOUT: LBI           DIGIT       ; POINT TO DIGIT POSITION
        LD            ; DIGIT POSITION TO A
        CAB           ; DIGIT POSITION TO BD
        OBD           ; OUTPUT DIGIT VALUE
        LEI           4           ; OUTPUT SEGMENT DATA (SET EN2)
        LBI           KBC         ; POINT TO KBC
        ING           ; G PORTS TO A
        AISC          1           ; ALL G PORTS STILL HIGH (= 15)?
        JMP           KEYDWN      ; NO, JUMP TO "KEYDOWN" ROUTINE
        CLRA
        AISC          3           ; YES, DELAY 13 INSTR. CYCLE TIMES
        JP            -1         ; BACK TO PREVIOUS INSTR. UNTIL SKIP
        LBI           KBC         ; POINT TO KBC

NRDY:  LDD           DIGIT       ; DIGIT POSITION TO A
        AISC          14         ; LAST DIGIT DONE (A = 1)?
        JMP           DEBOUN      ; YES, JUMP TO DEBOUNCE ROUTINE (A = 15)
        AISC          1           ; NO, DECREMENT DIGIT POSITION VALUE
        LBI           0,0        ; POINT TO DISPLAY REGISTER 0
        CAB           ; DIGIT POSITION VALUE TO BD

```

Figure 5.18 Display/Keyboard Interface Source Code



```

CLRA
AISC      4          ; DELAY 9 INSTR. TIMES
JP        -1        ; REPEAT PREVIOUS INSTR. UNTIL SKIP
JP        DSP2      ; DISPLAY NEXT DIGIT
.PAGE     1

; WORDS 0 - F EQUAL SEVEN-SEGMENT DECODE LOOKUP DATA TABLE
; I(7) - I(0) = SA - SG, D.P. SENT UPON LOOKUP TO Q(7) - Q(0), RESPECTIVELY
; HEX VALUE FOR CHARACTERS 0 - 9, P, A, U, C, F, E PLACED IN SUCCESSIVE LOCATIONS BY ".WORD" DIRECTIVE
.SPACE    5          ; LEAVE 5 BLANK LINES ON LISTING

.WORD     X'FD      ; = 0 (SEVEN-SEGMENT DECODE HEX VALUES)
.WORD     X'61      ; = 1
.WORD     X'DB      ; = 2
.WORD     X'F3      ; = 3
.WORD     X'67      ; = 4
.WORD     X'B7      ; = 5
.WORD     X'BF      ; = 6
.WORD     X'E1      ; = 7
.WORD     X'FF      ; = 8
.WORD     X'E7      ; = 9
.WORD     X'CF      ; = P
.WORD     X'EF      ; = A
.WORD     X'7F      ; = U
.WORD     X'90      ; = C
.WORD     X'8F      ; = F
.WORD     X'9F      ; = E

DEBOUN:   SKMBZ     3          ; UP BIT = 1?
           JP        ALLUP    ; YES
           SKMBZ     2          ; NO, NRB = 1?
           JP        STR      ; YES, A = 15 SO STORE IT IN KBC

DECKBC:   ADD       3          ; DECREMENT KBC
STR:      X         3          ; PLACE A IN KBC
           SMB       3          ; SET UP BIT OF KBC
           JMP       DSP1     ; DO DISPLAY LOOP OVER AGAIN

ALLUP:    SKMBZ     2          ; NRB = 1?
           JP        DECKBC   ; YES, DECREMENT KBC (A = 15)
           AISC      4          ; NO, SET KBC = 11
           NOP       4          ; DEFEAT "AISC" SKIP
           JP        STR      ; YES, RESTORE STRAP DIGIT VALUE

KEYDWN:   LDD       DIGIT     ; STRAP DIGIT POSITION TO A
           AISC      4          ; DIGIT POSITION > 11 (STRAP DATA)?
           JP        KBCTST   ; NO, TEST KBC
           AISC      12       ; YES, RESTORE STRAP DIGIT VALUE
           CAB       12       ; STRAP DIGIT POSITION TO BD
           CLRA
           AISC      1          ; 1 TO BR (POINT TO STRAP DATA REG. 1)
           XABR
           ING
           X         1          ; STRAP DATATO A
           JMP       NRDY     ; PLACE IN APPROPRIATE DIGIT, REG. 1

KBCTST:   RMB       3          ; RESET UP BIT OF KBC
           CLRA
           AISC      8          ; DELAY 5 INSTR. CYCLE TIMES
           JP        -1        ; REPEAT PREVIOUS INSTR. UNTIL SKIP
           CLRA          ; 0 TO A
           SKE
           JMP       NRDY     ; KBC = 0?
           LEI       0          ; NO
           ING        ; YES, BLANK SEGMENTS
           LBI       DIGIT     ; G PORTS TO A
           JMP       KEYDEC   ; POINT TO DIGIT NUMBER
           .FORM
           .PAGE     2          ; JUMP TO KEY DECODE ROUTINE
           .LIST     X'31      ; FORM FEED
           .INCLUD   TIMEKP    ; SUBROUTINE PAGE 2 CODE
           .PAGE     4          ; FULL MASTER LIST AND LIST OF INCLUDED "TIMEKP" CODE
           ; INCLUDE "TIMEKP" SUBROUTINE CODE

; FOLLOWING CODE USES CONTENTS OF A AND M, KEYSWITCH COLUMN AND ROW CLOSURE DATA, RESPECTIVELY, ON EXIT
; FROM DISPLAY ROUTINE, TO ACCESS ROM POINTERS TO JUMP TO KEY1 - KEY16 DECODE ROUTINES
; LABELS "KEY1" THROUGH "KEY16" MUST BE LOCATED WITHIN PAGES 4 THROUGH 7
.SPACE    5          ; FIVE BLANK LINES ON LISTING
KEYDEC:   COMP      5          ; COMPLEMENT A SO THAT BIT = 1 INDICATES KEY CLOSURE

```

Figure 5.18 Display/Keyboard Interface Source Code (continued)

```

JID          ; JUMP TO KEY DECODE ROUTINE FOR PARTICULAR KEY CLOSURE
.=          X'111 ; MOVE ASSEMBLER LOCATION COUNTER TO KEY1 ROM POINTER ADDRESS

ADDR        KEY1 ; PLACE KEY1 POINTER IN ADDRESS X'111
ADDR        KEY2 ; PLACE KEY2 - KEY4 POINTERS IN NEXT ROM LOCATIONS
ADDR        KEY3
ADDR        KEY4

.=          X'121 ; MOVE TO KEYS POINTER LOCATION
ADDR        KEY5
ADDR        KEY6
ADDR        KEY7
ADDR        KEY8
.=          X'141 ; MOVE TO KEY9 POINTER LOCATION (PAGE 5)
ADDR        KEY9
ADDR        KEY10
ADDR        KEY11
ADDR        KEY12
.=          X'181 ; MOVE TO KEY13 POINTER LOCATION (PAGE 5)
ADDR        KEY13
ADDR        KEY14
ADDR        KEY15
ADDR        KEY16

NODP: LBI     STORE ; POINT TO M(2,15)
      CQMA    ; SE - SG, D.P. TO A
      X       ; EXCHANGE INTO M(2,15)
      RMB     0 ; RESET D.P. BIT (DECIMAL POINT OFF)
      CAMQ    ; SEGMENT DATA BACK TO Q
      JMP     DIGOUT
  
```

Figure 5.18 Display/Keyboard Interface Source Code (continued)

Table 5.3 JID Pointer Table for Display/Keyboard Routine

Key No.	Value at Time of JID		Rows					Keyboard Columns			D0	JID Pointer Hex (X') Address	JID Pointer Hex Contents
	P9	P8	G3	G2	G1	G0	D3	D2	D1	M0			
1	0	1	0	0	0	1	0	0	0	1	X'111	85	
2	0	1	0	0	0	1	0	0	1	0	X'112	87	
3	0	1	0	0	0	1	0	0	1	1	X'113	89	
4	0	1	0	0	0	1	0	1	0	0	X'114	8B	
5	0	1	0	0	1	0	0	0	0	1	X'121	8D	
6	0	1	0	0	1	0	0	0	1	0	X'122	8F	
7	0	1	0	0	1	0	0	0	1	1	X'123	91	
8	0	1	0	0	1	0	0	1	0	0	X'124	93	
9	0	1	0	1	0	0	0	0	0	1	X'131	95	
10	0	1	0	1	0	0	0	0	1	0	X'132	97	
11	0	1	0	1	0	0	0	0	1	1	X'133	99	
12	0	1	0	1	0	0	0	1	0	0	X'134	9B	
13	0	1	1	0	0	0	0	0	0	1	X'141	9D	
14	0	1	1	0	0	0	0	0	1	0	X'142	9F	
15	0	1	1	0	0	0	0	0	1	1	X'143	A1	
16	0	1	1	0	0	0	0	1	0	0	X'144	A3	

COP CROSS ASSEMBLER  
COP420 DISPLAY

```

127                .FORM                ; FORM FEED

125                0100                .PAGE 4
126
127                ; FOLLOWING CODE USES CONTENTS OF A AND M, KEYSWITCH
128                ; COLUMN AND ROW CLOSURE DATA, RESPECTIVELY, ON EXIT FROM
129                ; DISPLAY ROUTINE, TO ACCESS ROM POINTERS TO JUMP TO
130                ; KEY1 - KEY16 DECODE ROUTINES. LABELS "KEY1" THROUGH
131                0005                .SPACE 5                ; "KEY16" MUST BE LOCATED WITHIN PAGES 4 THROUGH 7.
                                ; FIVE BLANK LINES ON LISTING

132 100 40 KEYDEC: COMP                ; COMPLEMENT A SO THAT BIT = 1 INDICATES KEY
133                ; CLOSURE
134 101 FF JID                ; JUMP TO KEY DECODE ROUTINE FOR PARTICULAR
135                ; CLOSURE
136                0111                = X'111                ; MOVE ASSEMBLER LOCATION COUNTER TO KEY1 ROM
137                ; POINTER ADDRESS
138 111 85 .ADDR KEY1                ; PLACE KEY1 POINTER IN ADDRESS
139 112 87 .ADDR KEY2                ; PLACE KEY2 - KEY4 POINTERS IN ROM LOCATIONS
140 113 89 .ADDR KEY3
141 114 8B .ADDR KEY4
142                0121                . = X'121                ; MOVE TO KEYS5 POINTER LOCATION
143 121 8D .ADDR KEY5
144 122 8F .ADDR KEY6
145 123 91 .ADDR KEY7
146 124 93 .ADDR KEY8
147                0141                . = X'141                ; MOVE TO KEY9 POINTER LOCATION (PAGE 5)
148 141 95 .ADDR KEY9
149 142 97 .ADDR KEY10
150 143 99 .ADDR KEY11
151 144 9B .ADDR KEY12
152                0181                . = X'181                ; MOVE TO KEY13 POINTER LOCATION (PAGE 6)
153 181 9D .ADDR KEY13
154 182 9F .ADDR KEY14
155 183 A1 .ADDR KEY15
156 184 A3 .ADDR KEY16

157 185 6002 KEY1: JMP DSPLY ; G0, D1 KEY
158 187 6002 KEY2: JMP DSPLY ; G0, D2 KEY
159 189 6002 KEY3: JMP DSPLY ; G0, D3 KEY
160 18B 6002 KEY4: JMP DSPLY ; G0, D4 KEY
161 18D 6002 KEY5: JMP DSPLY ; G1, D1 KEY
162 18F 6002 KEY6: JMP DSPLY ; G1, D2 KEY
163 191 6002 KEY7: JMP DSPLY ; G1, D3 KEY
164 193 6002 KEY8: JMP DSPLY ; G1, D4 KEY
165 195 6002 KEY9: JMP DSPLY ; G2, D1 KEY
166 197 6002 KEY10: JMP DSPLY ; G2, D2 KEY
167 199 6002 KEY11: JMP DSPLY ; G2, D3 KEY
168 19B 6002 KEY12: JMP DSPLY ; G2, D4 KEY
169 19D 6002 KEY13: JMP DSPLY ; G3, D1 KEY
170 19F 6002 KEY14: JMP DSPLY ; G3, D2 KEY
171 1A1 6002 KEY15: JMP DSPLY ; G3, D3 KEY
172 1A3 6002 KEY16: JMP DSPLY ; G3, D4 KEY

173 1A5 1D NODP: LBI STORE                ; POINT TO M(2,15)
174 1A6 332C CQMA                ; SE - SG, D.P. TO A
175 1A8 06 X                ; EXCHANGE INTO M(2,15)
176 1A9 4C RMB 0                ; RESET D.P. BIT (DECIMAL POINT)
177 1AA 333C CAMQ                ; SEGMENT DATA BACK TO Q
178 1AC 6019 JMP DIGOUT
179                ..END

```

Figure 5.19 Key Decode Routine — Output Listing



### 5.4 SIO Input/Output

SI and SO can be used to provide additional I/O capability for the COP400 family by connecting, for example, external 8-bit parallel-to-serial (MM74C165) and serial-to-parallel (MM74C164) shift registers, as shown in Figure 5.20. The following routine will output 8 bits of data serially using the SIO registers, at the same time inputting 8 bits serially. Data is output from and input to A and M. This program must be entered with the SIO register enabled as a serial shift register. The execution of an XAS instruction with C = "1" and "0" respectively will enable and disable SK as a SYNC output. (See Section 3.2, LEI instruction description.) With SK enabled as a SYNC output it will provide a clock pulse to the shift registers each instruction cycle time. Note that SI is simultaneously shifting 1 bit of serial data into SIO while SO is shifting 1 bit of serial data out. Since the 4-bit contents of SIO are continuously shifted each instruction cycle time, the routine is written to insure that SIO is exchanged with A every 4 instruction cycle times.

```

; ROUTINE TO OUTPUT 8 BITS OF DATA SERIALLY FROM M
; AND A WHILE INPUTTING 8 BITS OF SERIAL DATA INTO M
; AND A USING THE SIO REGISTER
; UPON ENTRY, SIO MUST BE ENABLED AS A SERIAL SHIFT
; REGISTER (ENO = 0)

```

```

SERIO:
SC      ; SET CARRY TO ENABLE SK AS A SYNC
        ; OUTPUT
XAS     ; START SYNC, A TO SIO, START SHIFTING
        ; A OUT, SI DATA IN
NOP     ; WAIT 4 INSTR. CYCLE TIMES
NOP
LD      ; M TO A
XAS     ; FIRST 4 SI BITS TO A, A TO SIO,
        ; CONTINUE SHIFTING SI IN, SO OUT
X      ; STORE FIRST 4 SI BITS IN M
CLRA   ; CLEAR A (WAIT 4 INSTR. CYCLE TIMES)
RC     ; RESET C TO DISABLE SK AS A SYNC
        ; OUTPUT
XAS     ; STOP SYNC, LAST 4 SI BITS TO A

```

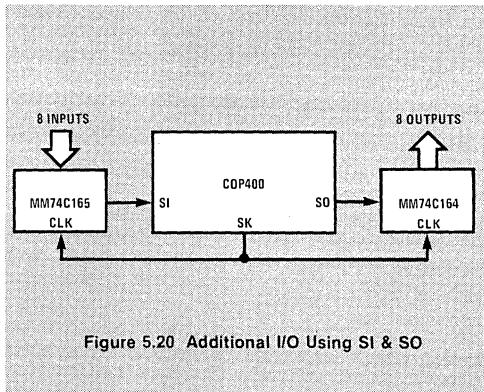


Figure 5.20 Additional I/O Using SI & SO

Figure 5.21 shows an example of a multi-COP420 system. As indicated, data transfers between the two devices are done in a serial fashion, with one COP providing a SYNC pulse via the SK output to the CKO pin of the second COP. To ensure the validity of the data being transferred, both COPs must contain a routine which will synchronize the inputting and outputting of data between the two devices using the SIO register. The following code accomplishes this by providing that each COP receive and send a string of four "1s" (SIO = 1111<sub>2</sub>) before an SIO data transfer is effected.

```

; ROUTINE TO SYNCHRONIZE SERIAL DATA TRANSFERS
; BETWEEN TWO COP DEVICES (COPA AND COPB) USING
; THE SIO REGISTER
; SIO MUST HAVE BEEN PREVIOUSLY ENABLED AS A SERIAL
; SHIFT REGISTER

```

```

; COPA CODE:
BACK:   NOP      ; ADD 1 INSTR. CYCLE TIME FOR
        ; RE-SYNC
        CLRA     ; ZERO TO A
        XAS     ; OUTPUT ZEROS, WAIT 4 INSTR.
        NOP     ; CYCLE TIMES
        CLRA
        COMP    ; 15 TO A
        XAS     ; OUTPUT 15 VIA SK, SI BITS TO A
        AISC 1  ; ARE INPUT BITS = 15?
        JP      BACK ; NO, TRY AGAIN
        .       ; YES, DEVICES SYNCHRONIZED
        .

```

```

; COPB CODE:
BACK:   CLRA     ; OUTPUT ZEROS IN 4-CYCLE
        ; LOOP
        XAS
        AISC 1  ; 15 FROM COPA?
        JP      BACK ; NO, KEEP SENDING OUT ZEROS
        COMP    ; YES, OUTPUT 15 TO COPA
        XAS
        NOP     ; DEVICES SYNCHRONIZED
        NOP
        NOP     ; WAIT FOR COPA TO START

```

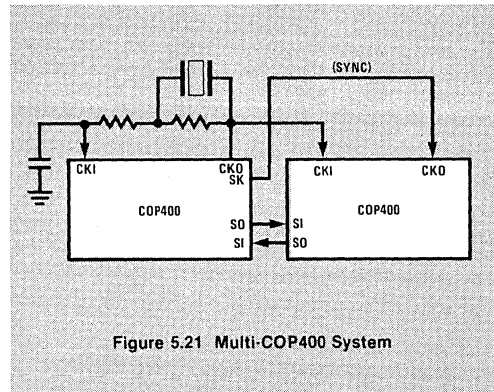


Figure 5.21 Multi-COP400 System

## 5.5 Add-On RAM

The following routine will interface the COP420 to an additional 2K bits (512 × 4) of RAM. The interconnect diagram (see Figure 5.22) shows the COP420 interfaced to two additional MM2112 (256 × 4) RAM devices, although CMOS equivalents (MM74C921s) may also be used where lower power consumption or RAM battery backup is desired. Up to four devices may be used by decoding the D<sub>0</sub> and D<sub>1</sub> lines (2-to-4 binary decoder). If all 4 bits of D are used, up to 16 additional RAM devices can be interfaced utilizing a 4-to-16 binary decoder (an additional 2K bytes of RAM).

The following routine treats the 1024 bits of external RAM as organized as 16 registers of 16 4-bit digits. It sequentially addresses digits 0 through 15 in a particular external RAM register (as determined by the 4-bit contents of COP RAM

memory digit M(3,15). It then reads from or write I/O data into COP RAM memory, register 0, digits 0-15, respectively.

Note that two different operands for the LEI instruction are used to select or de-select specific operations associated with three of the four bits of the EN register. The LEI 13 instruction sets EN<sub>3</sub>-EN<sub>0</sub> equal to 1101 with the result that EN<sub>3</sub> and EN<sub>0</sub> are equal to "1" and, therefore, SO will output a "1" to the WE pins of external RAM to perform a read operation. EN<sub>2</sub> is also set to "1" to enable the L drivers so that Q latch data will be output to the L I/O ports and, via the interconnect, to the RAM address lines. The LEI 5 instruction alters EN<sub>3</sub> to "0," resulting in SO being driven low, enabling a write operation into the external RAM device.

```
; SUBROUTINE TO READ FROM/WRITE TO ONE OF TWO EXTERNAL RAM DEVICES (256 x 4 BITS EACH)
; 16 4-BIT DIGITS OF I/O DATA READ FROM OR WRITTEN INTO COP RAM, REGISTER 0, DIGITS 0 - 15
; C=0 INDICATES A READ OPERATION, C=1 INDICATES A WRITE OPERATION
; 8-BIT RAM ADDRESS SPECIFIED BY A 4-BIT REGISTER NUMBER CONTAINED IN M(3,15), ASSIGNED TO SYMBOL "DIGIT"
; CHIP-SELECT NUMBER (1110 OR 1101 BINARY) CONTAINED IN M(2,15), ASSIGNED TO SYMBOL "CSEL"
; READ: ENTRY POINT TO READ RAM
; WRITE: ENTRY POINT TO WRITE RAM
```

```

      DIGIT      = 1,15
      CSEL       = 2,15
      REG        = 3,15

READ:
      RC          ; RESET CARRY FOR READ OPERATION
      JP          RW

WRITE:
      SC          ; SET CARRY FOR WRITE OPERATION
      RW:        ; READ/WRITE CODE
      OGI        15      ; SET G3 - G0 HIGH
      LEI        13      ; SO = 1, ENABLE L DRIVERS
      LBI        CSEL
      OBD        ; OUTPUT CHIP SELECT VALUE
      LBI        DIGIT   ; POINT TO DIGIT NUMBER
      CLRA       ; START WITH DIGIT 0

RWL:
      X          ; EXCHANGE A INTO DIGIT NUMBER IN M
      LDD        REG    ; REGISTER NUMBER TO A
      CAMQ       ; OUTPUT REGISTER AND DIGIT NUMBER FOR RAM ADDRESS
      LD         1      ; DIGIT NUMBER TO A, POINT TO REGISTER 0
      CAB        ; DIGIT NUMBER TO BD TO POINT TO I/O DATA IN M
      SKC        ; IS CARRY EQUAL TO 1?
      JP         RR     ; NO, JUMP TO READ RAM
      LEI        5      ; YES, PERFORM WRITE OPERATION, DRIVE WRITE ENABLE LOW
      OMG        ; OUTPUT DATA TO RAM
      LEI        13     ; SET WRITE ENABLE HIGH
      OGI        15     ; SET G3 - G0 HIGH

RWCONT:
      LBI        DIGIT  ; POINT TO DIGIT NUMBER
      LD         ; DIGIT NUMBER TO A
      AISC       1      ; INCREMENT DIGIT NUMBER, IS DIGIT = 15?
      JP         RWL   ; NO, CONTINUE READ/WRITE
      OBD        ; YES, DISABLE RAMS (CHIP SELECTS HIGH)
      RET        ; RETURN

RR:
      ING        ; READ RAM DATA
      X          ; STORE IN I/O DIGIT IN M
      JP         RWCONT ; CONTINUE
```

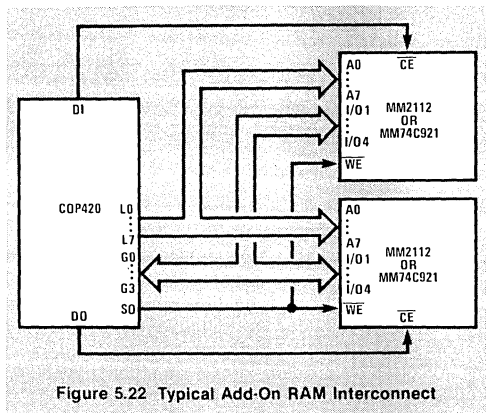


Figure 5.22 Typical Add-On RAM Interconnect

### 5.6 IN<sub>3</sub>/IN<sub>0</sub> Inputs

Section 4.8 has already provided an example of an interrupt service routine utilizing the "hardware" interrupt capability of the IN<sub>1</sub> COP420 pin. It is also possible to implement a "software" interrupt, using either the COP420 IN<sub>3</sub> or IN<sub>0</sub> inputs, since they

have testable input latches associated with them. These latches, IL<sub>3</sub> and IL<sub>0</sub>, will be set if a low going pulse, at least two instruction cycles wide, has occurred on the IN<sub>3</sub> or IN<sub>0</sub> inputs, respectively. The INIL instruction inputs these latches to A, as explained in Section 3.2, to allow them to be tested as software interrupt flags (A<sub>3</sub> and A<sub>0</sub>).

To accomplish a software interrupt, an INIL instruction must be executed often enough to respond to the requirements of the interrupt signal tied to IN<sub>3</sub> or IN<sub>0</sub>. For example, in timekeeping applications, IN<sub>3</sub> or IN<sub>0</sub> may be connected to a 60 Hz square wave. The program must, in this case, execute an INIL instruction at least every 1/60 second.

If an interrupt input occurs irregularly, it will be more efficient to connect it to the hardware interrupt pin, IN<sub>1</sub>, to insure that no interrupt is missed due to infrequent testing. Conversely, if an interrupt input occurs regularly and predictably (such as a 60 Hz signal) a software interrupt may be efficiently utilized by simply building into the program a sufficient test rate to insure that no inputs are missed.

### Technical Assistance

National Semiconductor will be pleased to provide technical assistance to aid a user in design and development. Inquiries may be directed to any of our Field Applications Engineers (FAEs) — located in every National sales office — or to our in-plant COPS™ Applications Group at (408) 737-5582.

# Analog to Digital Conversion Techniques With COPS™ Family Microcontrollers

National Semiconductor  
Leonard A. Distaso  
February 1980  
COP Note 1



## Table of Contents

I. Introduction .....	2-84
II. Simple Capacitor Charge Time Measurement .....	2-84
A. Basic Approach .....	2-84
B. Accuracy Improvements .....	2-87
C. Conclusions .....	2-88
III. Pulse Width Modulation (Duty Cycle) Technique .....	2-89
A. Mathematical Analysis .....	2-89
B. Basic Implementation .....	2-90
C. Accuracy Improvements .....	2-93
IV. Dual Slope Integration Techniques .....	2-97
A. Mathematical Background .....	2-97
B. Basic Dual Slope Technique .....	2-98
C. Modified Dual Slope Technique .....	2-100
V. Voltage to Frequency Converters, VCO's .....	2-103
A. Basic Approach .....	2-103
B. The LM131/LM231/LM331 .....	2-105
C. Voltage Controlled Oscillators .....	2-105
D. A Combined Approach .....	2-105
VI. Successive Approximation .....	2-107
A. Basic Approach .....	2-107
B. Some Comments on Resistor Ladders .....	2-109
VII. "Offboard" Techniques .....	2-112
A. General Comments .....	2-112
B. ADC0800 Interface .....	2-112
C. Naked-8™ Interface .....	2-113
D. The MM5407 as an A/D Converter .....	2-115
VIII. Conclusion .....	2-118
IX. References .....	2-118

## I. Introduction

A variety of techniques for performing analog to digital conversion are presented. The COP420 microcontroller is used as the control element in all cases. However, any of the COPS™ family of microcontrollers could be used with only minor changes in some component values to allow for different instruction cycle times.

All indirect analog to digital converters are composed of three basic building blocks:

- D/A Converter
- Comparator
- Control logic

In a software driven system the D/A converter and comparator are present but the control logic is replaced by instruction sequences. There are a variety of software/hardware techniques for implementing A/D converters. They differ primarily in their approach to the included D/A. There are two primary approaches to the digital to analog conversion which can in turn be divided into a number of sub-categories:

- D/A as a function of weighted closures
  - R/2R ladder
  - Binary weighted ladder
- D/A as a function of time
  - RC exponential charge
  - Linear charge/discharge (dual slope)
  - Pulse width modulation

These techniques should be generally familiar to persons skilled in the electronic art. The objective here is to illustrate the application of these established methods to a low cost system with a COPS microcontroller as the intelligent control element. Circuit configurations are provided as well as the appropriate flow charts and code to implement the function.

Some mathematical and theoretical analysis is presented as an aid to understanding the various techniques and their limits. However, it is not the purpose here to provide a definitive theoretical analysis of the analog to digital conversion process or of the various techniques described.

## II. Simple Capacitor Charge Time Measurement

### A. BASIC APPROACH

#### A.1 General

Perhaps the simplest means to perform an analog to digital conversion is to charge a capacitor until the capacitor voltage is equal to the unknown voltage. The capacitor voltage and the unknown are compared by means of a standard analog comparator. The unknown is determined simply by counting, in the microcontroller, the amount of time it takes for the charge on the capacitor to reach a value equal to the unknown voltage. The capacitor voltage is given by the standard capacitor charge equation:

$$V_C = V_0 + [V_1 - V_0][1 - e^{-(t/RC)}]$$

where:  $V_C$  = capacitor voltage  
 $V_0$  = "discharge voltage" — low level voltage  
 $V_1$  = high level voltage

The most obvious problem with this method, from the standpoint of software implementation, is the nonlinearity of the relationship. This can be circumvented in

several ways. First of all, a routine to calculate the exponential can be implemented. This, however, usually requires too much code if the exponential routine is not otherwise required in the program. Alternatively, the range of input voltages can be restricted so that only a portion of the capacitor charge curve — which can be approximated with a linear relationship or with some minor straight line curve fitting — is used. Finally, a look up table can be used which will effectively convert the measured time to the appropriate voltage. The look up table has the advantage that all the math can be built into the table, thereby simplifying matters significantly. If arithmetic routines are going to be used, it is clear that the relationship is simplified if  $V_0$  is 0 volts because it then drops out of the equation.

## A.2. Basic Circuit Implementation

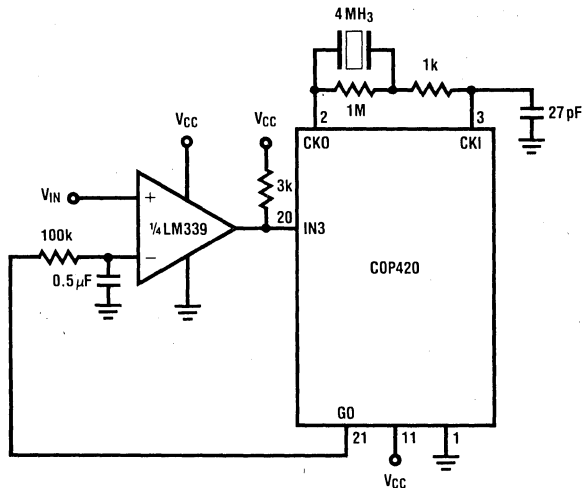
The circuit in Figure 1 is the basic implementation of the capacitor charge method of A/D conversion. The selection of input and output used is arbitrary and is dictated by general system considerations.  $V_0$  is the "0" level of the G output and  $V_1$  is the "1" level of the output. The technique is basically to discharge the capacitor to  $V_0$  (which is ideally ground) and then to apply  $V_1$  and increment an internal counter until the comparator changes state. The flow chart and code for this implementation are shown in Figure 2.

## A.3 Accuracy Considerations

The levels reached by the microcontroller output constitute one of the more significant problems with

this basic implementation. The levels of  $V_1$  and  $V_0$  are not  $V_{CC}$  and ground as would be desired. The level is defined by the load on the output, the value of  $V_{CC}$ , and the device itself. Furthermore, these levels are likely to change from device to device and over temperature. To be sure, the output values will be at least those given in the data sheet, but it must be remembered that those values are minimum high voltages and maximum low voltages. Typically, the high value will be greater than the spec minimum and the low value will be lower than the spec maximum. In fact, with a light load the values will be close to  $V_{CC}$  and ground. Therefore, in order to obtain any accurate result for a voltage measurement the exact values of  $V_1$  and  $V_0$  need to be measured and somehow stored in the microcontroller. Typical values of these voltages can be measured experimentally and an average could be used for a final implementation.

The other problem associated with the levels is that the capacitive load on the output line is substantial and far in excess of the values used when specifying the characteristics of the various COP420 outputs. The significant effect of this is that it will take longer than "normal" for the output to reach its maximum value. In addition, it is likely that there will be dips in the output as it rises to its maximum value since the capacitor will start to draw charging current from the output. All of this will be fast relative to the other system times. Still, it will affect the result since the level to which the capacitor is attempting to charge is not being applied uniformly and "instantaneously". It can be viewed as though the voltage  $V_1$  is bouncing before it stabilizes.



CRYSTAL OSCILLATOR VALUES CHOSEN TO GIVE  $4\mu\text{s}$  CYCLE TIME WITH DIVIDE BY 16 OPTION  
SELECTED ON COP 420 CK0/CKI PINS

$V_{CC} = +5V$

Figure 1. Basic Capacitor Charge Technique



```

DCI      0      ;TURN OFF G TO DISCHARGE CAPACITOR
; INSERT SOME DELAY TO MAKE SURE CAPACITOR DISCHARGED
; USING 12 BIT COUNTER, BUT ONLY UPPER 8 USED IN TABLE
; LOOK UP DUE TO ACCURACY OF RC CHARGE METHOD. THE OTHER
; BITS COULD BE USED BUT THE COMPLICATIONS ARE NOT WORTH
; THE EFFORT FOR THIS PARTICULAR TECHNIQUE. ALSO, HERE THE
; INPUT RANGE IS RESTRICTED SO THAT THE TOP 3 BITS ARE ZERO

RCAD:    DCI      1      ;TURN ON THE G LINE
INCR:    LBI      2,13   ;BINARY INCREMENT OF 12 BIT COUNTER
BINPLI:  SC       ;LOWER FOUR BITS WILL BE DISCARDED
BINPLI:  CLRA     ;ONLY TOP BITS USED IN TABLE LOOK UP
ASC      ;SPEED WOULD BE IMPROVED IF THE ADD WERE
NOP      ;STRAIGHT LINE CODED--BUT COSTS MORE CODE
XIS
JP
BINPLI
ININ     ;READ IN3 TO SEE IF COMPARATOR CHANGED
AISC     8
JP      END
CLRA
JP      INCR
END:     DCI      0      ;TURN OFF THE G LINE AND DISCHARGE C
; DO ARITHMETIC HERE OR LOOK UP TABLE OR WHATEVER IS
; REQUIRED--SAMPLE LOOK UP TABLE CONTROL INDICATED BELOW
; SAMPLE TABLE WRITTEN CORRECTING FOR THE EXPONENTIAL
; RELATIONSHIP. THE TABLE ALSO INCORPORATES A CONVERSION
; TO BCD. THE VALUE IN THE TABLE IS THE RATIO OF
; THE CAPACITOR VOLTAGE V TO THE MAXIMUM VOLTAGE VMAX.
; THE NUMBER IS A TWO DIGIT BCD FRACTION. WE ARE USING
; A 5 BIT COUNT IN THIS EXAMPLE. ADDRESSING ARBITRARILY
; SET UP ASSUMING THAT CONTROL CODE IS IN PAGE 0 (OTHER
; THAN AT ADDRESS 0) AND THAT THE TABLE THEREFORE IS IN
; PAGE 1 (STARTING AT HEX ADDRESS 040).
;
LBI      2,15   ;POINT TO TOP 4 BITS
XDS      ;TOP 4 IN A, POINTING TO LOWER 4 IN 2,14
AISC     4      ;THIS MERELY ADJUSTING FOR ADDRESS--NO
; OTHER FUNCTION
LQID     ;DO THE LOOK UP
CGMA     ;FETCH THE ADJUSTED VALUE FROM G
; THE ADJUSTED VALUE IS NOW IN A AND M. FROM THIS POINT MAY
; USE THE VALUE IN OTHER CALCULATIONS OR OUTPUT THE INFORMATION,
; OR WHATEVER MAY BE REQUIRED BY THE APPLICATION.
LBI      2,13   ;CLEAR THE COUNTER
STII     0
STII     0
STII     0
JP      RCAD:   ;JUMP BACK AND REPEAT

;=X'040 ;SET UP TABLE ADDRESS
WORD 000,003,006,008 ;SET UP THE TABLE VALUES
WORD 011,014,016,019 ;HERE, COMPENSATED FOR EXPONENTIAL
WORD 021,023,026,028 ;AND CONVERTED TO BCD FRACTION
WORD 030,032,034,036 ;TABLE VALUE IS RATIO V/VMAX
WORD 038,039,041,043
WORD 045,046,048,049
WORD 051,052,053,055
WORD 056,057,059,060

```

Figure 2A. Typical RC Charge A/D Code

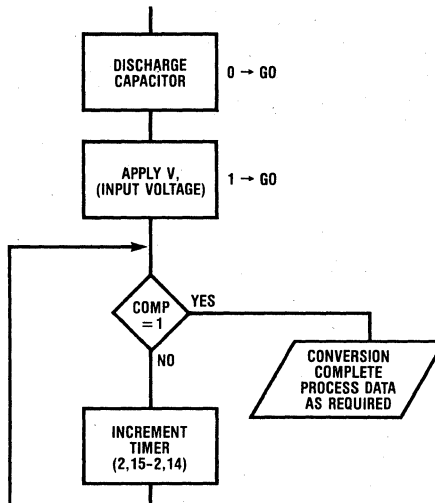


Figure 2B. RC Charge Flow Chart

A more general problem is that of the tolerance of RC time constant. The value of the voltage with respect to time is obviously related to the RC value. Therefore, a change in that value will result in a change in the voltage for a given time period  $t$ . The graph in Figure 3 illustrates the effect of a  $\pm 10\%$  variation in the RC value upon the voltage measured for a given time  $t$ . If one cares to work out the math, it comes out that the error is an exponential relationship in much the same manner as the capacitor voltage itself. The maximum error induced for  $\pm 10\%$  RC variation is  $\pm 3.9\%$ .

Remember also that we are measuring time. Therefore variation in the RC value will have a direct, linear effect on the time required to measure a given voltage. It is also necessary that the time base for the COP420 be accurate. A variation in the accuracy in the operating frequency of the COP420 will have a direct impact on the accuracy of the result.

Given the errors mentioned so far and assuming that no changes are made in the hardware, the accuracy of the technique then is determined by the resolution of the time measurement. This is improved in two ways: increase the RC time constant so that there is a smaller change in capacitor voltage for a given time period or try to minimize the loop time required to increment the counter. Lengthening the RC time constant is easier but the cost is increased conversion time. The minimum time to increment a 5 to 8 bit binary counter and test an

input is 13 cycle times. For a 9 to 12 bit binary counter this minimum time is 17 cycle times. Note also that the minimum time to perform the function does not necessarily correspond to the minimum number of code words required to implement the function. At a cycle time of 4 microseconds, the 13 cycle times correspond to 52 microseconds.

## B. ACCURACY IMPROVEMENTS

Several options are available if it is desired to improve the accuracy of this method. Three such improvements are shown in Figure 4. Figure 4A is the smallest change. Here a pullup resistor has been added to the G output line and the G line is run open drain internally, i.e., the internal pullup is removed. This improves the "bounce" problem mentioned earlier. The G line will go to the high state and remain there with this setup. However, the addition of the resistor does little more than eliminate the bounce. The degree of improvement is not great, but it is an easy way to eliminate a minor source of error.

Figure 4B is the next step. A 74C04 is used as a buffer. The 74C04 was chosen because of its symmetric output characteristics. Any CMOS gate with such characteristics could be used. The software can easily be adjusted to provide the proper polarity. The COP420 output drives a CMOS gate which in turn drives the RC network. This change does make significant improvements in accuracy. With a light load the CMOS gate will typically

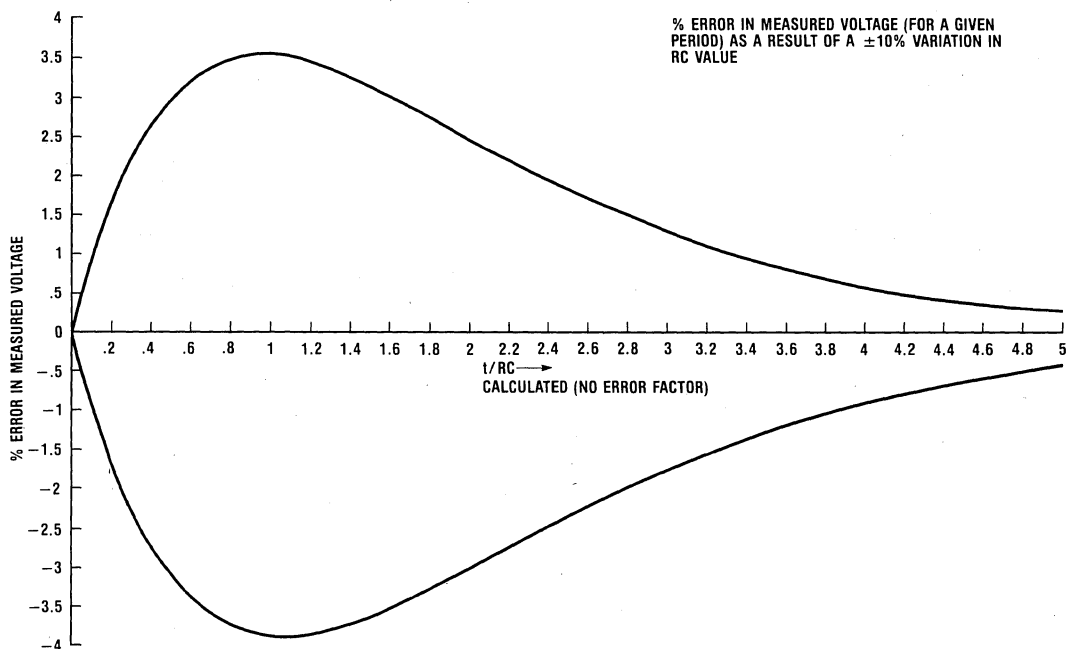


Figure 3

swing from ground to  $V_{CC}$  and its output level is not as likely to be affected by the capacitor discharge.

Figure 4C is the best approach, but it involves the greatest component cost. Here two G outputs are controlling analog switches. Ground is connected to the RC network to discharge the capacitor, and a positive reference is used to charge the capacitor. This reference can be any suitable voltage source: zener diodes,  $V_{CC}$ , etc. The controlling voltage tolerance is now clearly the tolerance of the reference. Precise voltage references are readily obtainable. Figure 4C also shows an analog switch connected directly across the capacitor to speed up the capacitor discharge time. When using this version of the basic scheme, remember to include the 'on' resistance of the analog switch connected to  $V_{REF}$  in the RC calculation. Failure to do so will introduce error into the result.

Note that the LM339 is a quad comparator. If these comparators are not otherwise needed in the system, they can be used in much the same manner as the CMOS gate mentioned above. They can be used to buffer the output of the COPSTM device and to reset the capacitor, or whatever other function is required. This has the advantage of fully utilizing the components in

the system and eliminates the need to add another package to the system.

### C. CONCLUSIONS

This approach is an inexpensive way to perform an A/D conversion. However, it is not that accurate. With a 10%  $V_{CC}$  supply and a 10% tolerance in the RC value and 10% variation in the oscillator frequency the best that can be hoped for is about 25% accuracy. If a 1% reference voltage is used, this accuracy becomes about 15%.

Under laboratory conditions — holding all variables constant and using precise measured values in the calculations — the configuration of Figure 2 yielded 5 bit accuracy over an input range of 0 to 3.5 volts. Over the same range and under the same conditions, the circuit of Figure 4B yield 7 to 8 bit accuracy. It must be emphasized that these accuracies were obtained under controlled conditions. All variables were held constant and actual measured values were used in all calculations. It is unlikely that the general situation will yield these accuracies unless adjustments are provided and a calibration procedure is used. This could defeat the low cost objective.

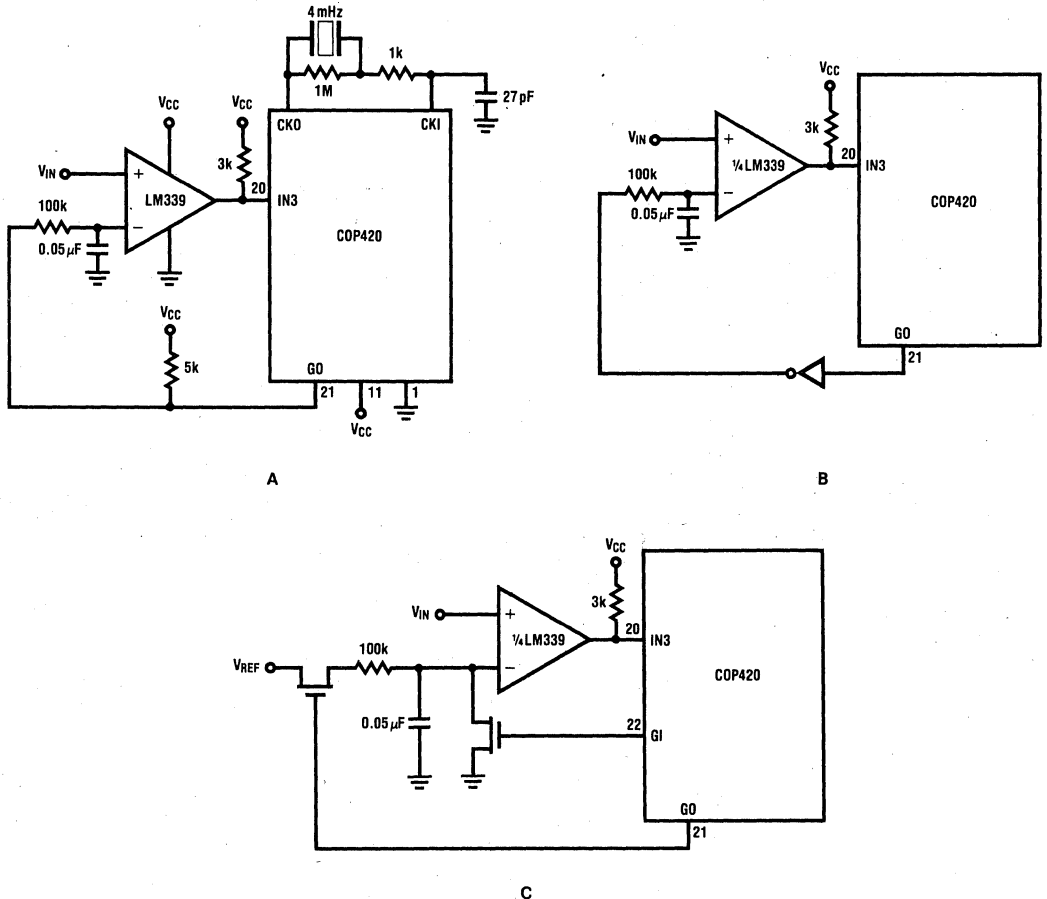


Figure 4

### III. Pulse Width Modulation (Duty Cycle) Technique

#### A. MATHEMATICAL ANALYSIS

The pulse width modulation, or duty cycle, conversion technique is based on the fact that if a repetitive pulse waveform is applied to an RC network, the capacitor will charge to the average voltage of the waveform provided that the RC time constant is sufficiently large relative to the pulse period. See Figure 5.

In this technique, the capacitor voltage  $V_C$  is compared to the voltage to be measured by means of an analog comparator. The duty cycle is then adjusted to cause  $V_C$  to approach the input voltage. The COPSTM device reads the comparator output and then drives one of its outputs high or low depending on the result, i.e., if  $V_C$  is lower than the input voltage, a positive voltage ( $V_1$ ) is applied to charge the capacitor; if  $V_C$  is higher than the input voltage, a lower voltage ( $V_0$ ) is applied to discharge the capacitor. Thus the capacitor voltage will seek a point where it varies above and below the input voltage by a small amount. Figure 6 illustrates the capacitor voltage and the comparator output.

Some mathematical analysis here will be useful to help clarify the technique and to point out its restrictions. Referring to Figure 6, we have the following:

$$V_A = V_0 + [V_B - V_0][e^{-(t_1/RC)}]$$

$$V_B = V_A + [V_1 - V_A][1 - e^{-(t_2/RC)}]$$

$$= V_1 + [V_A - V_1][e^{-(t_2/RC)}]$$

solving for  $t_1$  and  $t_2$  we have:

$$t_1 = -RC \ln[(V_A - V_0)/(V_B - V_0)]$$

$$t_2 = -RC \ln[(V_B - V_1)/(V_A - V_1)]$$

let:

$$V_A = V_{IN} - d_1$$

$$V_B = V_{IN} + d_2$$

substituting the above, the equations for  $t_1$  and  $t_2$  become:

$$t_1 = -RC \ln\{[1 - (d_1/(V_{IN} - V_0))]/[1 + (d_2/(V_{IN} - V_0))]\}$$

$$t_2 = -RC \ln\{[1 - (d_2/(V_{IN} - V_1))]/[1 - (d_1/(V_{IN} - V_1))]\}$$

the equations reduce by means of the following assumptions:

1.  $d_1 = d_2 = d$
2.  $|V_{IN} - V_0| \gg d$   
 $|V_{IN} - V_1| \gg d$

applying these assumptions, we get the following:

$$t_1 = -RC \ln[(1+x)/(1-x)] \text{ where } x = -d/(V_{IN} - V_0)$$

$$t_2 = -RC \ln[(1+x)/(1-y)] \text{ where } y = d/(V_{IN} - V_1)$$

because of the assumptions above, the  $x$  and  $y$  terms in the preceding equations are less than 1, therefore the following expansion can be used:

$$\ln[(1+z)/(1-z)] = 2[z + (z^3)/3 + (z^5)/5 + \dots]$$

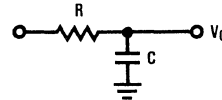
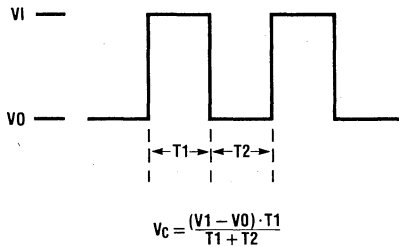


Figure 5

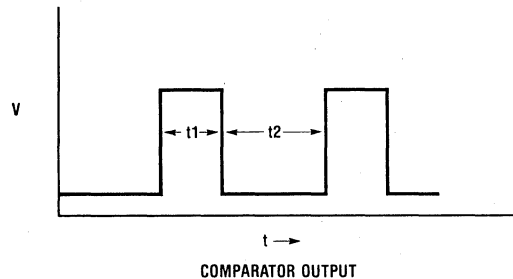
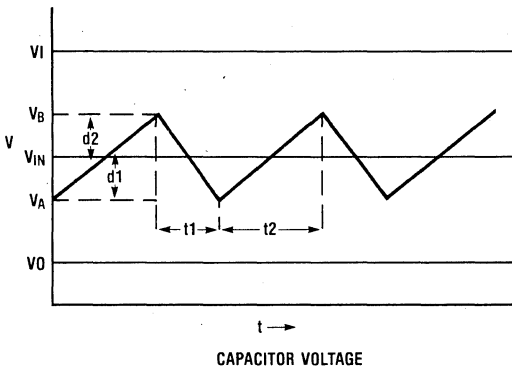


Figure 6

substituting we have:

$$t1 = -2RC[x + (x^*3)/3 + \dots]$$

$$t2 = -2RC[y + (y^*3)/3 + \dots]$$

under assumption 2 above, the linear term completely swamps the exponential terms yielding the following result (after substituting back into the equation):

$$t1 = 2dRC/(V_{IN} - V0) \quad t2 = -2dRC/(V_{IN} - V1)$$

therefore:

$$t1/(t1 + t2) = (V1 - V_{IN})/(V1 - V0)$$

$$t2/(t1 + t2) = (V_{IN} - V0)/(V1 - V0)$$

solving for  $V_{IN}$ :

$$V_{IN} = [t2/(t1 + t2)][V1 - V0] + V0$$

$$\text{or } V_{IN} = V1 - [t1/(t1 + t2)][V1 - V0]$$

It follows from the above results that by measuring the times  $t1$  and  $t2$ , the input voltage can be accurately determined. As will be seen, the restrictions based upon the assumptions above do not cause any serious difficulty.

## A.2 General Accuracy Considerations

In the preceding calculations it was assumed that the differential output above and below the input voltage was the same. If the comparator output is checked at absolutely regular intervals, and if the intervals are kept as small as possible this assumption can be fairly easily guaranteed — at least to within the comparator offset which is only a few millivolts. As we shall see, this aspect of the technique presents few, if any, difficulties. In addition, there is an RC network at the input of the comparator. The time constant of this network must be long relative to the time between checks of the comparator output. This will insure that the capacitor voltage does not change very much between checks and thereby help to insure that the differences above and below the input voltage are the same.

The next major approximation has to do with the difference between the input voltage and either  $V1$  or  $V0$ . We have relied on this difference being much greater than the amount the capacitor voltage changes above and below the input voltage. This approximation allows the nonlinear terms in the logarithmic expansion to be discarded. In practicality, the approximation means that the input voltage must not be "close" to either  $V1$  or  $V0$ . Therefore, it becomes necessary to determine how closely the input voltage can approach  $V1$  or  $V0$ . It is obvious that the smaller the difference  $d$  can be made, the closer the input voltage can approach either reference. The following calculations illustrate the method for determining that difference  $d$ . Note, using either  $V1$  or  $V0$  produces the same result. Thus  $V = V1 = V0$ .

For at least 1% accuracy

$$x + (x^*3)/3 < 1.01x$$

$$\text{therefore } x < 0.173$$

$$\text{since } x = d/[(V_{IN} - V)] \text{ we have } d < 0.173 [(V_{IN} - V)].$$

Using the same analysis for 0.1% accuracy in the approximation we get  $d < 0.0548[(V_{IN} - V)]$ . By applying this relationship, the RC time constant can be adjusted so that, within the time interval, the capacitor voltage does not change by more than  $d$  volts. The user may

then select, within reason, how close to the references he can allow the input voltage to go.

The next consideration is really just one of simplification. It is clear that if  $V0$  is zero, it drops out of the first equation and the relationship is simplified. Therefore, it is desirable to use zero volts as the  $V0$  value. The equation then becomes:

$$V_{IN} = V1t2/(t1 + t2).$$

It is obvious by now that the heart of the technique lies in accurately measuring the times  $t1$  and  $t2$ . Clearly this requires that the time base of the COP420 be accurate. Short term variations in the COP420 time base will clearly impact the accuracy of the result. In addition to that there is a serious problem in being able to check the comparator output often enough to get any accuracy and resolution out of simply measuring the times  $t1$  and  $t2$ . This problem is circumvented by measuring many periods of the waveform. Doing this gives a large average, which improves the accuracy and tends to eliminate any spurious changes. Of course, the trade off is increased time to do the conversion. However if the time is available, the technique becomes restricted only by the accuracy of the external components. Those of the comparator and the reference voltage are most critical.

It is clear from the equation above that the accuracy of the result is directly dependent upon the accuracy of the reference voltage  $V1$ . In other words, it is not possible to be more accurate than the reference voltage. If, however, all that is required is a ratio between the input voltage and the reference voltage, the accuracy of the reference will not be a controlling factor provided that the input voltage tracks the reference. This requires that the input voltage be generated from the reference voltage in some form, e.g., a voltage divider with  $V_{IN}$  coming off a variable resistance.

Finally, we have noted that the difference  $d$  must be small. If the capacitor had to charge or discharge a long way toward  $V_{IN}$ , the nonlinearity of the capacitor charge curve would be significant. This therefore requires that the conversion begin with the capacitor voltage close to the input voltage.

Note that the RC value is not part of the equation. Therefore the accuracy of the time constant has no effect on the result as long as the time constant is long relative to the time between checks of the comparator output.

The final point is that the reference voltages, whatever they may be, must be hard sources. Should these voltages vary or drift at all, they will directly affect the result. In those configurations where the references are being switched in and out, the voltage should not change when it is switched into the circuit.

## B. BASIC IMPLEMENTATION

### B.1. General

The objective, then, is to measure the times  $t1$  and  $t2$ . This is accomplished in the software by means of two counters. One of the two counters counts the  $t2$  time; the other counter counts the total time  $t1 + t2$ .

It is necessary to check the comparator output at regular intervals. Thus the software must insure that

path lengths through the test and increment loops are equal in time. Further it is desirable to keep the time required to increment the counters as short as possible. A trade off usually comes into play here. The shortest loop in terms of code required to implement the function is rarely the shortest loop in terms of time required to execute the function. The user has to decide which implementation is best for him. The choice will frequently be governed by factors other than the A/D conversion limits.

It must be remembered that we are now dealing with analog signals. If significant accuracy is required, we are handling very small analog signals. This requires the user to take precautions that are normally required when working with linear circuits, e.g., power supply decoupling and bypassing, lead length restrictions, crosstalk, op amp and comparator stabilization and compensation, desired and undesired feedback, etc. As greater accuracy is sought these factors are more and more significant. It is suggested that the reader refer to the National Semiconductor Linear Applications Handbook and to the data sheets for the various components involved to see what specific precautions should be taken both in general and for a specific device.

## B.2 The Basic Circuit

Figure 7 shows the diagram for the basic circuit required to implement the duty cycle conversion scheme. The flow chart and code required to implement the function are shown in Figure 8. Note that the flow chart and code do not change — except for possible polarity change on output to allow for an inverting buffer — for any of the improvements in accuracy discussed later. The only exception to this is the technique illustrated in Figure 10 and the variations there are minor.

The code and flow chart in Figure 8 implement the technique as described above. The large averaging technique is used as it would be too difficult to measure the times  $t_1$  and  $t_2$  in a single period. The total time,  $t_1 + t_2$ , is the viewing window under complete control of the software. This window is a time equal to the total number of counts, determined by desired accuracy, multiplied by the loop time for a single count. A second counter is counting the  $t_2$  time. Special care is taken to insure that all paths through the code take the same length of time since the integrity of the time count is the essence of the technique. The full conversion scheme would use the subroutine in Figure 8. Normally the subroutine would be called first just to get the capacitor charged close to the input voltage. The result obtained here would be discarded. Then the routine would be called a second time and the result used as required.

In the configuration in Figure 7, there is an RC network in both input legs of the comparator. This is to balance the inputs of the device. For this reason,  $R_1 = R_2$ .  $C_1$  is the capacitor whose voltage is being varied by the pulse waveform.  $C_2$  is in the circuit only for stabilization and symmetry and is not significant in the result. The comparator tends to oscillate when the + and - inputs are nearly equal without capacitor  $C_2$  in the circuit.

As would be expected, the basic circuit has some difficulties. By far the most serious of these difficulties is the output level of the G line. To be sure of the high and low level of this output the levels should be measured. The "1" level will be between the spec minimum of 2.4V and  $V_{CC}$  (here assumed to be 5 volts). The "0" level will be between the 0.4V spec maximum and ground. With light loads, these levels are likely to vary from device to device. Furthermore, we have the same "1" level problem that was mentioned in the simplest technique: the capacitive load is large and the capacitor is

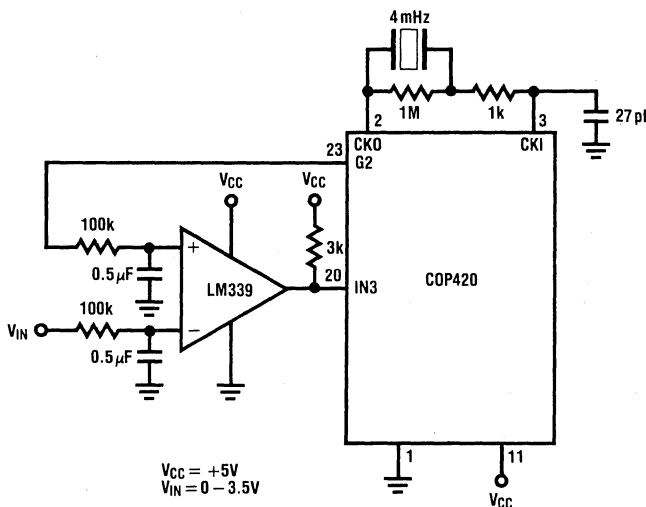


Figure 7. Basic Duty Cycle A/D

charging while the output is trying to go to the high level.

There is also a problem with the low level. When the output goes low, the capacitor begins to discharge through the output device of the COP420. This discharge current has the effect of raising the "0" level and thereby introducing error. Note that we are not talking about large changes in the voltages, especially the low level. Typically, the change will only be a few millivolts but that can translate into a loss of accuracy of several bits.

Under laboratory conditions — holding all variables constant and using precise measured values in the calculations — the circuit of Figure 7 yielded 5 bit  $\pm 1$  bit accuracy over the range of  $V_0$  (here measured to be 0.028 volts) to 3.5 volts (the maximum specified input voltage for the comparator with  $V_S = 5$  volts). Increasing the number of total counts had very little effect on the result. In the general case, the basic scheme should not be relied upon for more than 4 bits of accuracy, especially if one assumes that  $V_1 = V_{CC}$  and  $V_0 = 0$ . As shall be seen, it is not difficult to improve this accuracy considerably.

```

;A100) IS THE FULL CONVERSION SCHEME WRITTEN AS A SUBROUTINE
A100:  LBI    1,10    ;MAKE SURE COUNTERS CLEARED
        JSRP   CLEAR
        LBI    2,10
        JSRP   CLEAR
        LBI    1,13    ;PRELOAD FOR TOTAL COUNT = 2048
        STII   0
        STII   0
        STII   8
A1001:  ININ                    ;READ COMPARATOR--INPUT TO 420 = IN3
        AISC   8
        JP    SND01
SND01:  LBI    3,0    ;USING DMG BELOW TO SAVE STATE OF OTHER 0
        ;VALUES IF IT WAS NECESSARY TO DO SO, ELSE USE OGI
        SMB    2    ;VIN > Vc, DRIVE Vc HIGHER
        DMG                    ;THIS CODE STRAIGHT LINED FOR SPEED
        SC                    ;APPLY POSITIVE REFERENCE
        CLRA                    ;INCREMENT THE SUB COUNTER
        LBI    2,13
        ASC
        NOP
        XIS
        CLRA
        ASC
        NOP                    ;BINARY INCREMENT
        XIS                    ;WOULD ELIMINATE THESE 4 WORDS IF 8 BIT
        CLRA                    ;COUNTER OR LESS--HERE SET UP FOR UP TO 12 BIT
        ASC                    ;COUNTER
        NOP
        X
        JP    TOTAL
SND01:  LBI    3,0
        RMB    2
        DMG
        CLRA
        AISC   10    ;THIS PART OF THE CODE MERELY INSURES THAT
        NOP        ;ALL PATHS THROUGH THE ROUTINE ARE EQUAL IN TI
DI Y:   AISC   1
        JP    DLY
TOTAL:  CLRA
        LBI    1,13
        SC
        ASC        ;INCREMENT THE TOTAL LOOP COUNTER
        NOP        ;WHEN OVERFLOW, DONE SO EXIT
        XIS
        CLRA
        ASC
        NOP
        XIS
        CLRA
        ASC
        JP    ATOD2
        RET
A1002:  X
        JP    ATOD1
        .PAGE 2
CLEAR:  CLRA
        XIS
        JP    CLEAR
        RET

```

Figure 8A. Duty Cycle A/D Code

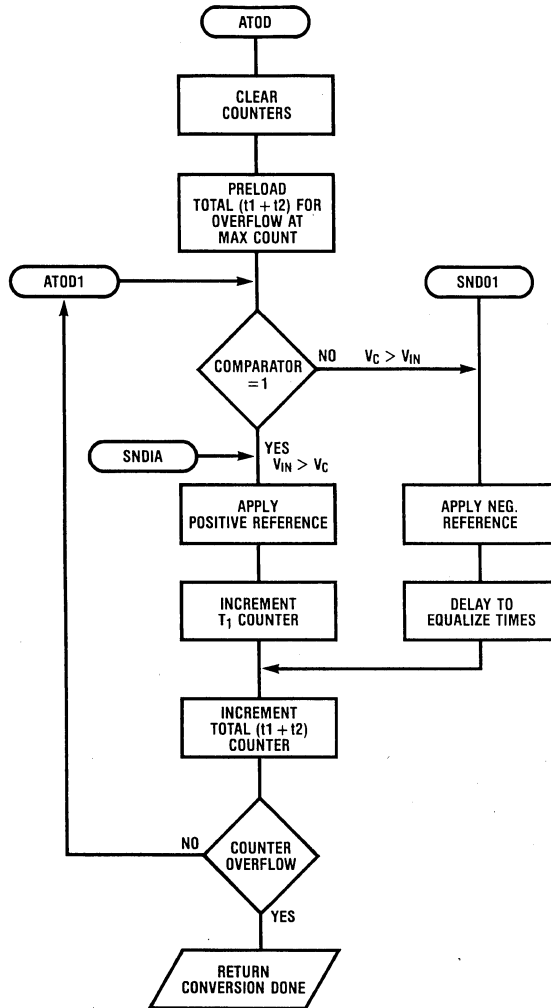


Figure 8B. Duty Cycle A/D Flow Chart

## C. ACCURACY IMPROVEMENTS

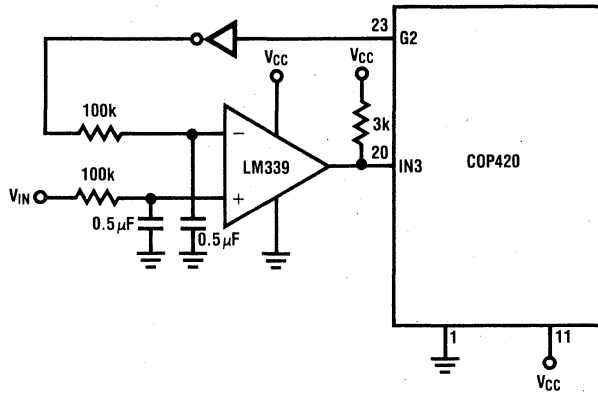
### C.1 General Improvements

Figure 9 illustrates circuit changes that will make significant improvements in the accuracy of the technique. In Figure 9A a CMOS buffer is used to drive the RC network. The output of the COP420 drives the CMOS gate, which here is a 74C04 because of its output characteristics. The main thing that this technique does is to reduce the difficulties with the output levels. Typically,  $V_0$  is 0 volts and  $V_1$  is  $V_{CC}$ . We also have a "harder" source for the voltages — the levels don't change while the capacitor is charging or discharging. Now, even more clearly than before, the accuracy of  $V_{CC}$  is the controlling voltage tolerance. The accuracy of the result will be no better than the accuracy of  $V_{CC}$  (for a system requiring absolute accuracy).

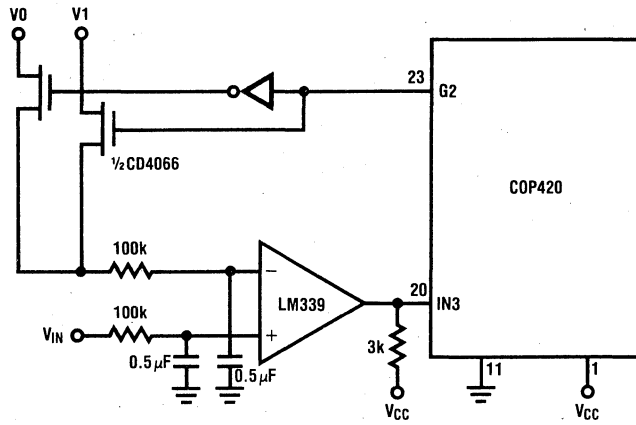
Under laboratory conditions, the circuit of Figure 9A yielded the accuracies as indicated below for various total counts. The accuracy increased with the total count until the count exceeded 2048. There was no significant increase in accuracy with this circuit for counts in excess of 2048. (Remember that these results were obtained under controlled conditions). We may then view the results obtained with 2048 counts as the upper limit of accuracy with the circuit of Figure 9A. The results were as follows:

Total Count	Resultant Accuracy
512	$8 \pm 1/2$ bits
1024	$9 \pm 1$ bits
2048	$9 \pm 1/2$ bits
4096	$9 \pm 1/2$ bits

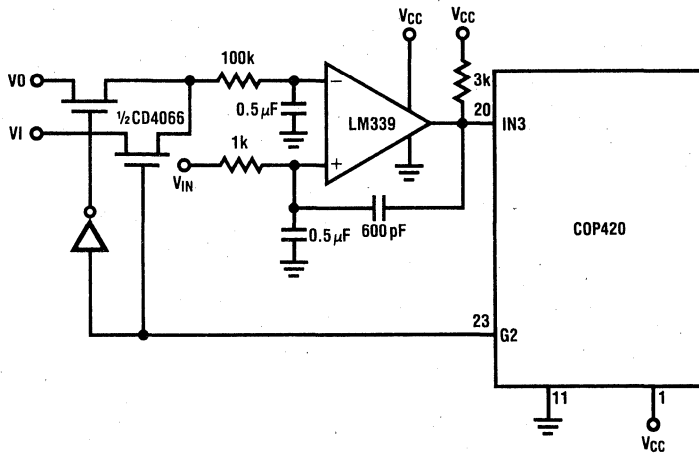




A



B



C

Figure 9. Improvements to Duty Cycle A/D

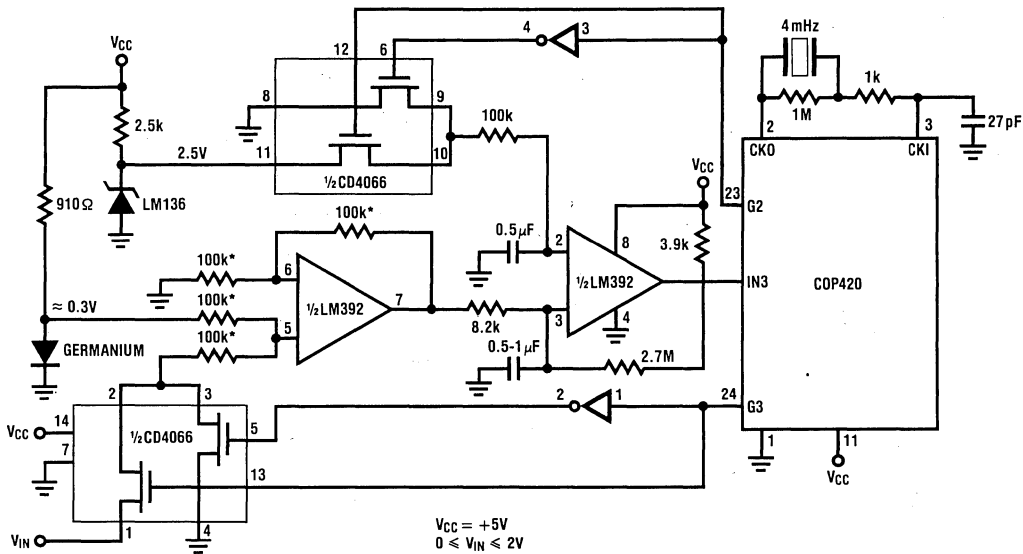
The circuit of Figure 9B makes a significant change to improve accuracy. Now the COP420 is controlling analog switches and switching in positive and negative references. Therefore the accuracy of the reference voltages is the controlling factor. Generally this will improve the accuracy over that obtained with Figure 9A. With the circuit of Figure 9B, with  $V_0 = 1$  volt (negative reference), and  $V_1 = 3$  volts (positive reference), 9 bit accuracy was achieved with a total count of 1024.  $V_0$  and  $V_1$  were arbitrarily chosen to place the input voltage approximately in the center of the allowable comparator input range with  $V_S = 5$  volts. Remember, the accuracy of the references is controlling. The result can be no more accurate than the references. Furthermore, these references must be hard sources; i.e., they must not change when they are switched into the circuit as that contributes error into the result.

In Figure 9C, capacitive feedback was added to the comparator circuit and the series resistance to  $V_{IN}$  was decreased. The feedback added hysteresis and forced the comparator to slew at its maximum rate (significant errors are introduced if the comparator does not change state in a time shorter than the cycle time of the controller). Both of these changes resulted in increased accuracy of the result. With  $V_0 = 0$ ,  $V_1 = 5$  volts ( $V_{CC}$ ) and  $V_{CC}$  held steady at 5.000 volts, an accuracy of 10 bits  $\pm 1$  bit was achieved over the input range of 0 to 3.5 volts.

It is obviously possible to use any combination of the configurations in Figure 9 for a given application. What is used will depend on the user and his specific requirements.

Figure 10 illustrates a further refinement of the basic approach. This configuration can be used if greater accuracies are needed. The major change is the addition of a summing amplifier to the circuit for the purpose of adding a fixed offset voltage to the input voltage. This has the effect of moving the input voltage away from the negative reference (which is 0 volts here). This offset voltage should be stable as the changes in it will directly affect the result. The offset voltage should be chosen so as to place the effective input voltage (the voltage at the comparator input) approximately in the center of the range between the two references. The precise value of the offset is not critical nor is its source. The forward voltage drop across a germanium diode is used as the offset in Figure 10, but this offset can be generated in any convenient manner. The forward voltage drop of the germanium diode is approximately 0.3 volts. Given this and the negative reference of 0 volts and a positive reference of 2.5 volts, the input voltage is restricted to a range of 0 to 2 volts. Therefore, the effective input voltage (at the comparator input) is approximately 0.3 volts to 2.3 volts — well within the limits of the two references. The circuit also includes provision for an autozero self calibration procedure.

Note that the resistors in the summing amplifier should be matched. The absolute accuracy of these resistors is not significant, but their accuracy relative to one another can have a significant bearing on the result. The restriction is imposed so that the output of the summing amplifier is exactly the sum of the input voltage and the offset voltage. This requires unity gain



\*RESISTORS SHOULD BE MATCHED

Figure 10. Improved Duty Cycle A/D with Autozero

through the amplifier and that the impedance in each summing leg be the same. These effects can become very serious if one is trying for significant accuracy — e.g., if 12 bit accuracy is being sought 1% matching of those resistors can introduce an error of 1% maximum. While 1% accurate is fairly good, it is significantly less than 12 bit accuracy. Related to this effect is a possible problem with the source impedance of the input voltage. If that impedance is significant in terms of its ratio to the summing resistor, errors are introduced just as if the resistors are mismatched. "Significant" is determined in terms of the desired system accuracy and the relative impedance values. The comparator section is using some feedback to provide hysteresis for stability and a low series resistance is used for the input to the comparator.

Most significantly, this configuration allows a true zeroing of the system. Through the additional analog switches shown, the COP420 can easily perform an

autozero function by tying the input to ground and measuring the result. Thus the system offsets can be calculated, stored and subtracted from the result. This improves the accuracy and is also more forgiving on the choice of the comparator and op amp selected. Furthermore, the offset can be periodically recomputed by the COP420 thereby compensating for drift in system offsets. Nonetheless, the accuracy of the reference is the controlling factor. It is NOT possible to obtain an absolute (as opposed to ratiometric) accuracy of 12 bits without a reference that is accurate to 12 bits. The LM136 used in Figure 10 is a 1% reference. Although not inherently accurate to 12 bits, the voltage of the LM136 may be trimmed to an exact value by means of a variable resistor. The data sheet of the LM136 illustrates this connection. Under laboratory conditions, the circuit of Figure 1 yielded 11 bit  $\pm 1$  bit accuracy with a total count of 4096 over the input range of 0 to 2 volts. Figure 11 indicates the flow chart and the code required to implement the technique of Figure 10.

```

;CODE FOR IMPROVED A TO D PULSE WIDTH METHOD
;SEE FIGURE 8A FOR CODE FOR ROUTINE ATOD
;
AUTZER: LBI    3,0    ;DO AUTO ZERO,3,0 CONTAINS G STATUS
        RMB    3      ;SET UP TO GRND INPUT & MEASURE OFFSET
        JSR    ATOD   ;FIRST TIME IS TO GET CLOSE
        JSR    ATOD   ;MEASURE THE OFFSET
        LBI    2,13   ;NOW SAVE THE OFFSET VOLTAGE
XIFR:   LD     1      ;SAVE THE OFFSET VALUE IN M3
        XIS    1
        JP     XFER
        LBI    0,0
        JP     INPUT
MEASUR: ;NOW DO REAL MEASUR(1ST TIME IS OFFSET)
        JSR    ATOD   ;FIRST TIME TO GET CLOSE
        JSR    ATOD   ;NOW REAL MEASUREMENT
        JSRP   BINSUB ;SUBTRACT THE OFFSET
;HAVE THE VALUE AT THIS POINT(IN BINARY)—NOW DO WHAT
;THE APPLICATION REQUIRES. VALUE MUST BE MULTIPLIED
;BY (VREF+/TOTAL COUNT) TO GET FINAL VALUE IF SUCH IS
;DESIRED
        LBI    1,0    ;INCREMENT COUNTER FOR NEW OFFSET MEASURE
        LD     1
        AISC   1
        JP     SAVE
        X      ;IS 16TH TIME, MEASURE OFFSET AGAIN
        JP     AUTZER
SAVE:   X
        LBI    3,0
        SMB    3      ;SET BIT SO CAN MEASURE VIN
        JP     MEASUR
        PAGE   2
BINSUB: LBI    3,13
        SC
BNSUB2: LD     1
        CASC
        NDP
        XIS    1
        JP     BNSUB2
        RET

```

Figure 11A. Duty Cycle A to D, Improved Method

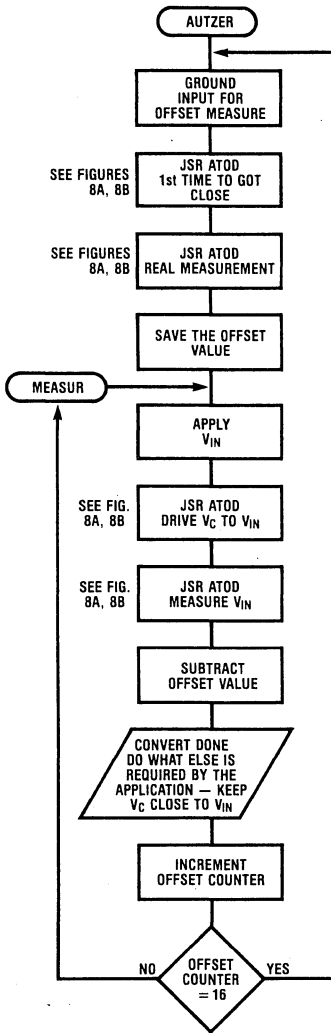


Figure 11B. Flow Chart for Improved Duty Cycle A/D

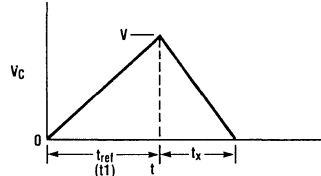
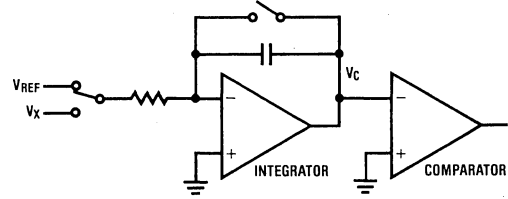


Figure 12. Dual Slope Integration — Basic Concept

$$I_x = C \frac{dV}{dt} = V_x/R$$

$$V_x = RC \frac{dV}{dt}$$

$$\int_0^{T1} V_x dt = \int_0^{T1} RC dV$$

$$V_x T1 = RC V$$

$$V = V_x T1 / RC = I_x T1 / C$$

Similarly:

$$I_{REF} = C \frac{dV}{dt} = V_{REF} / R$$

$$V_{REF} = RC \frac{dV}{dt}$$

$$\int_{T1}^{T1+Tx} V_{REF} dt = \int_V^0 RC dV$$

$$V_{REF} T_x = -RC V$$

$$V = -V_{REF} T_x / RC$$

$$-V_{REF} T_x / RC = V_x T1 / RC$$

$$V_x = -V_{REF} T_x / T1$$

Two important facts arise from the preceding mathematics. First of all, there is a linear relationship involved in determining the unknown voltage. Secondly, the negative sign in the final equation indicates that the reference and the unknown, relative to some point (which may be 0 volts or some bias voltage), have opposite polarity. Thus, if it is desired to measure 0 to +5 volts, the reference voltage must be -5 volts. If the input is restricted to 2.5 to 5 volts, the reference can be 0 volts as the integrator and comparator are biased at +2.5 volts (then the 0 volts is in fact -2.5 volts relative to the biasing voltage, and the input range is 0 to 2.5 volts relative to the same bias voltage).

There are some difficulties with dual polarity conversion using the dual slope method. It is clear from the math above that if the input voltage will be dual polarity, it is necessary to have two references — one of each polarity. The midrange biasing arrangement briefly

## IV. Dual Slope Integration Techniques

### A. MATHEMATICAL BACKGROUND

(Some of this background information is taken from National Semiconductor Linear Applications Note AN-155. The reader is referred to that document for other related general information.)

The basic approach of dual slope integration conversion techniques is to integrate a voltage across a capacitor for a fixed time, and then to integrate in the other direction with a known voltage until the starting point is reached. The ratio of the two times then represents the unknown voltage. Some of the math below in conjunction with Figure 12 will illustrate the approach.

described above eliminates the need for two different polarities but does not help very much since two references are still required — one at the positive value and one at the bias value. Ground is the other reference. Further, the need to select one of two references further complicates the circuitry involved to implement the approach. Also, the dual requirement brings up a difficulty with the bias currents of the integrator and comparator. They could add to the slope in one polarity and subtract in the other.

The only real operational difficulty in dual slope systems is establishing the initial conditions on the integrating capacitor. If this capacitor is not at the proper initial conditions, accuracy will be severely impaired. Figure 12 indicates a switch across the capacitor as a means of initializing it. In a software driven system, the initialization can be accomplished by doing two successive conversions. The result of the first conversion is discarded. It is performed only to initialize the capacitor. The second conversion produces the valid result. One need only insure that there is not significant time lapse between the two conversions. They should take place immediately after one another.

This approach obviously lengthens conversion time but it eliminates many problems. The alternative to this approach of two successive conversions is to take a great deal of care in insuring the initial state of the integrating capacitor and in selecting op amps and comparators with low offsets.

**B. THE BASIC DUAL SLOPE TECHNIQUE**

Figure 13 indicates an implementation of the basic dual slope technique. This is a single polarity system and thus requires only the single reference voltage. The circuit of Figure 13 is perhaps not the cheapest way to implement such a scheme but it is representative and illustrates the factors that must be considered.

Consider first the means of initializing the integrating capacitor C1. The routine here connects the input to ground and does a conversion on zero volts as a means of initialization. Subsequently — and this is typical of the more usual technique — two conversions are performed. The first conversion is to initialize the capacitor. The second conversion yields the result. Some form of initialization or calibration procedure is required to achieve optimum accuracy from dual slope conversion schemes.

The comparator in this circuit is used in the inverting mode and has positive feedback as recommended in the LM111 data sheet. The voltage reference is the LH0070, which is a 0.01% reference. A resistive voltage divider on the LH0070 creates the 5 volt value. The use of the voltage divider brings up two difficulties (which can be overcome if the LH0070 is used at its full value, thus eliminating the divider, and the result properly scaled in the microcontroller or series integrating resistor increased). First, the impedance of the reference must be small relative to the series resistance used in the integrator. If this were not the case, the

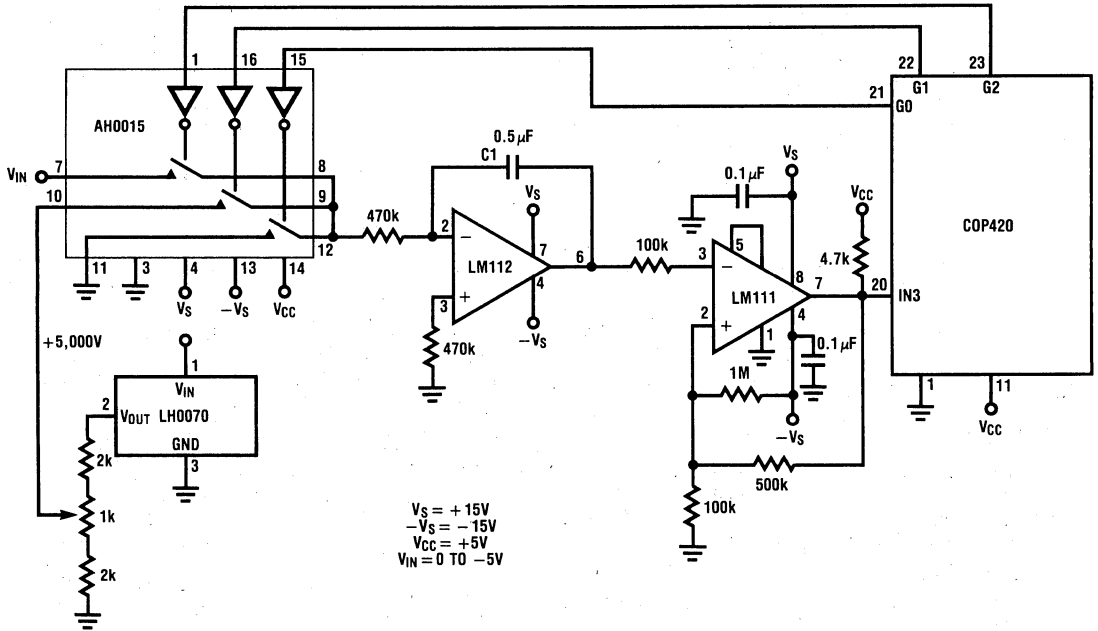


Figure 13. Basic Dual Slope Integration A/D Scheme

slopes would show an effect due to the difference in the R value between the applied reference voltage and the unknown input. (By the same token, the output impedance of the source supplying the unknown must also be small relative to that series integrating resistor). Secondly, the bias currents of the integrator may be such as to affect the reference voltage when it is coming from a simple resistor divider. Both problems are reduced if small resistor values are used in the divider. Note also that current mode switching would reduce the problem as well. It should be pointed out that the errors introduced by these problems are not gross deviations from the expected value. They are small errors that will not make much difference in the majority of applications. They are, however the kind of errors that can make the difference between a system accurate to 10 bits and one accurate to 12 bits (assuming all other factors the same).

Figure 14 shows the flow chart and code required to implement the basic dual slope technique as shown in Figure 13. Under laboratory conditions an accuracy of 12 bits  $\pm 1$  bit was achieved. The method is slow, with the maximum conversion time equal to  $2 \times T_{REF}$ . Notice that the accuracy of  $V_{CC}$  and that of the integrating resistor and capacitor are not involved in the accuracy of the result. The accuracy of  $V_{REF}$  is, of course, controlling if absolute accuracy — rather than ratiometric accuracy — is desired. The absolute accuracy of the circuit can be no better than the accuracy of the reference. If ratiometric accuracy is all that is required, there is no particular problem. The accuracy is merely relative to the reference. The R and C values do not impact the accuracy because the integration in both directions is being done through the same R and C. Results would be quite different if a different value of R or C was used for one of the slopes.

```

DUALSLOPE: OGI      1      ; HOLD THE INPUT TO GROUND TO RESET THE
LBI        2, 11     ; INTEGRATING CAPACITOR
JSRP      CLEAR     ; CLEAR THE COUNTER
JSR       INCRA     ; TO GET US CLOSE, NEXT READING IS REAL
CLEAR?:   LBI        2, 11     ; NOW CLEAR THE COUNTER
JSRP      CLEAR     ; MAKE SURE COUNTER CLEARED TO ZERO
; ; 15 = 0 AND START AT 1, 13 FOR COUNT = 4096
; ; 15 = 14 AND START AT 1, 12 FOR COUNT = 8192
; ; 15 = 12 AND START AT 1, 12 FOR COUNT = 16384
; FOLLOW SAME PATTERN FOR OTHER COUNTS
;
MEASURE:  JSR       INCRA     ; RUN THRU THE INCREMENTS
; NOW HAVE THE BINARY VALUE, USE IT AS IS OR
; MULTIPLY BY (Vref/TOTAL COUNT) TO CREATE THE VOLTAGE
; RESULT--THEN CONTINUE WITH THE OPERATION
LBI        2, 11
JSRP      CLEAR     ; CLEAR THE COUNTER
JSR       INCRA     ; TO GET CAP CLOSE TO 0 AGAIN
JP        CLEAR2
; FOLLOWING SUBROUTINE INCRA IS THE REAL PART OF THE ROUTINE
; CONCERNED WITH THE COUNTING FOR THE CONVERSION.
INCRA:    LBI        1, 15     ; R1 IS CLEARED PRIOR TO START
STII      15        ; PRESET THE COUNTER FOR 4096
OGI       4          ; APPLY VIN
INCR:     LBI        1, 12
SC
BINAD1:   CLRA
ASC
NOP
XIS
JP        BINAD1
NOP      ; 2 NOPS TO EQUALIZE TIMES
NOP
SKC
JP        INCR
OGI       2          ; DONE, NOW APPLY VREF
INCR?:    LBI        2, 12     ; COUNT UNTIL COMPARATOR CHANGES
SC
BINAD2:   CLRA
ASC
NOP
XIS
JP        BINAD2 ; STRAIGHT LINE THE ADD FOR SPEED
; SAVE WORDS BY USING G
AISC      8          ; SEE IF IN3=1
JP        INCR2     ; IN3 IS 0, KEEP COUNTING
OUTPUT:   OGI       1          ; KEEP INPUT AT 0
RET

```

Figure 14A. Dual Slope A/D Code

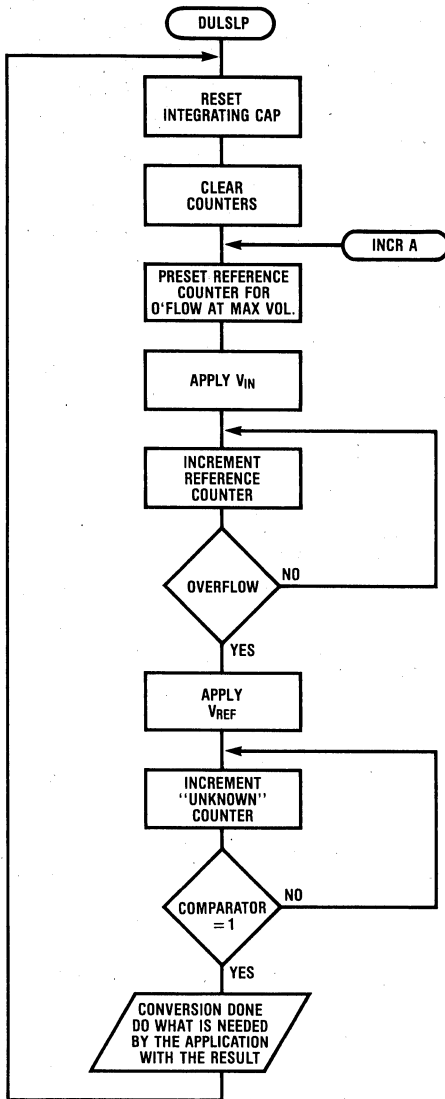


Figure 14B. Basic Dual Slope A/D Flow Chart

### C. MODIFIED DUAL SLOPE TECHNIQUE

#### C.1 General

The basic idea of the modified dual slope technique is the same as that of the basic approach. The modified approach eliminates the need for dual polarity references and is also more forgiving in the selection of the op amp and comparator required. Figure 15 illustrates the basic idea.

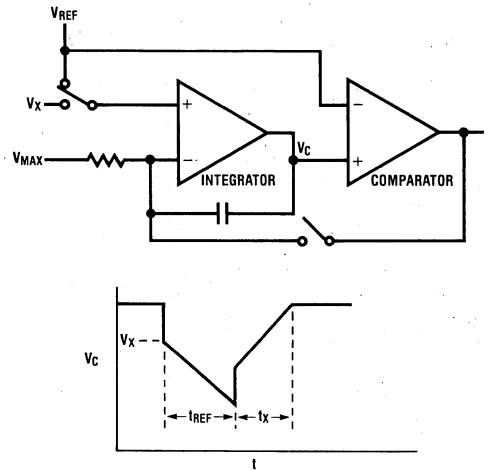


Figure 15. Modified Dual Slope — Basic Concept

The math analysis is much the same:

$$I_X = C \frac{dV}{dt} = (V_X - V_{MAX})/R$$

$$V_X - V_{MAX} = RC \frac{dV}{dt}$$

$$(V_X - V_{MAX})T_1 = RC$$

$$V = (V_X - V_{MAX})T_1/RC$$

Similarly:

$$I_{REF} = C \frac{dV}{dt} = (V_{REF} - V_{MAX})/R$$

$$(V_{REF} - V_{MAX})T_X = -VRC$$

$$V = -(V_{REF} - V_{MAX})T_X/RC$$

$$(V_{MAX} - V_{REF})T_X = (V_X - V_{MAX})T_1$$

$$V_X = V_{MAX} + (V_{MAX} - V_{REF})T_X/T_1$$

The main difference between this and the basic approach is the offset voltage  $V_{MAX}$ . The main restriction is that all input voltage values ( $V_X$ ) are less than  $V_{MAX}$ . It is also apparent that the total count is proportional to the difference between  $V_{MAX}$  and  $V_X$ . The only significant effect of this is, however, to slightly complicate the arithmetic required to arrive at a value for  $V_X$ .

Given that the input voltage  $V_X$  is always less than  $V_{MAX}$ , the modified dual slope technique is automatic polarity. This fact comes straight out of the equation above. Thus dual polarity references are not required. However, two precise voltages are required:  $V_{MAX}$  and  $V_{REF}$ . However, the  $V_{MAX}$  value can be used for a zero adjust as indicated in Figure 16. This means that the  $V_{MAX}$  value need not be so precise as it will be adjusted in a calibration procedure to produce a zero output. This adjustment amounts to a compensation for the bias currents and offsets. Thus the COP420 can use the supposed value of  $V_{MAX}$  with  $V_{MAX}$  later being "tweaked" to give the proper result at zero input. In addition, the initialization loop for the integrating capacitor includes the comparator. Thus the initial condition on the capacitor becomes not zero but the

sum of the offset voltages of the comparator and op amp. Thus the choice of these components is not critical in a modified dual slope approach.

## C.2 An Example of the Modified Dual Slope Approach

Figure 16 illustrates an implementation of the modified dual slope technique. The system is calibrated by holding  $V_{IN}$  to ground and then adjusting  $V_{MAX}$  for a "0" result. Capacitor C1 is the integrating capacitor. Capacitor C2 is used only to cause a rapid transition on the comparator output. C2 is especially useful if an op amp is being used as the comparator stage. Resistor R1 is just part of the capacitor initializing loop. An LH0070 is being used to generate the reference voltage and the  $V_{MAX}$  value. The discussion previously about these being hard sources is equally relevant here. In fact, this problem was much more significant in this particular implementation and made the difference between a 10 and 12 bit system. As shown, the technique was accurate to 10 bits. Another bit was obtained when the  $V_{MAX}$  and  $V_{REF}$  values were buffered. It must be remembered that when trying to achieve accuracies of this magnitude board layout, parts placement, lead length, etc. become significant factors that must be specifically addressed by the user.

There are some other considerations in using this technique. The amount of time required to count the specified number of counts starts to become a significant factor. If it takes "too long" to do the counting, the

capacitor can charge to either supply voltage depending on which direction it is integrating. This causes the wave shape shown in Figure 15 to flatten out. This effectively limits the input range for all accuracy is lost once that waveform flattens out. In fact, this was the limiting factor on the accuracy in Figure 16 as shown. Given the amount of time required for an increment of the counter for  $T_{REF}$  (or  $T_X$ ), it was not possible to reach the 4096 counts required for 12 bit accuracy before the waveform flattened out. Decreasing the total count solves the problem at the expense of accuracy. It is therefore desirable to keep the loop time required for an increment as fast as possible. The code to implement Figure 16 is shown in Figure 17 and reflects that concern. The other way to solve the problem is to use a large value for R and C. This is the easiest solution and preserves accuracy. Its cost is increased conversion time.

Both the basic and modified dual slope schemes can be very accurate and are commonly used. They tend to be relatively slow. In many applications, however, speed is not a factor and these approaches can serve very well. There are various approaches to dual slope analog to digital conversion which try to improve speed and/or accuracy. These are usually multiple ramping schemes of one form or another. The heart of the approach is the basic scheme described above. It is not the purpose here to delve into all the possible ways that dual slope conversion may be accomplished. The control software is not significantly different regardless of which particular variation is used. The basic ramping control is the same as that indicated here.

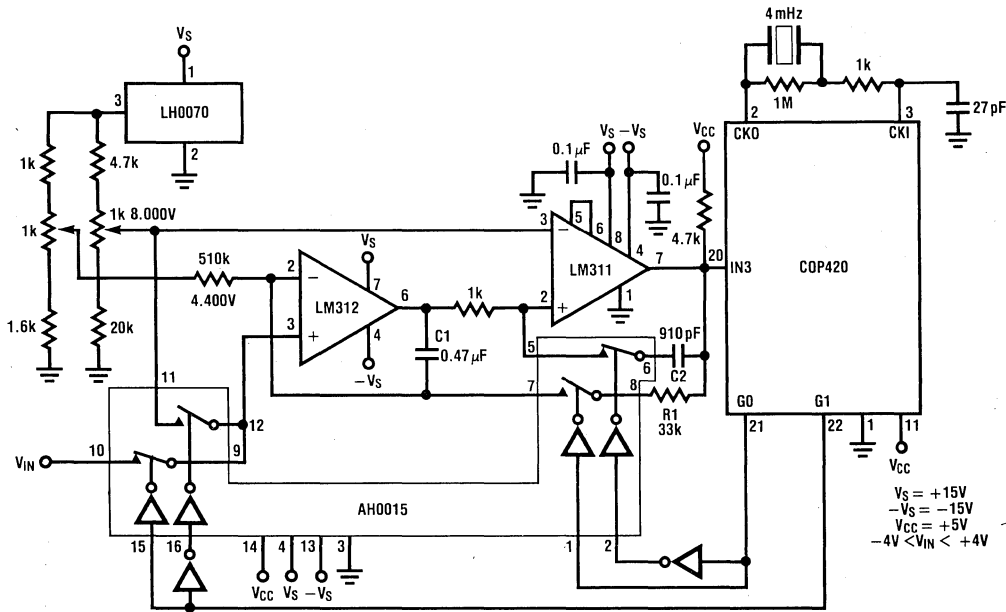


Figure 16. Modified Dual Slope Integration



The number of components required to implement a dual slope scheme is not related to the desired accuracy. The approach is generally tolerant as to the op amps and comparators used as long as proper care is given to the initialization of the integrating capacitor.

Precise references are not required if a ratiometric system is all that is required. Cheaper switches can be safely used. The dual slope scheme controlled by a COPS™ microcontroller can be a very cost effective solution to an analog to digital conversion problem.

```

CIFAR2: DGI      1      ; APPLY VREF AND ENABLE RESET PATH
        LBI      2, 11  ; NOW CLEAR THE COUNTER
        JSRP     CLEAR
        ; 1, 15=15, 1, 14=4 AND START AT 1, 12 FOR COUNT = 3072
        ; 1, 15 =15 AND START AT 1, 12 FOR COUNT = 4096
        ; 1, 15 = 14 AND START AT 1, 12 FOR COUNT = 8192
        ; 1, 15 = 12 AND START AT 1, 12 FOR COUNT = 16384
        ; FOLLOW SAME PATTERN FOR OTHER COUNTS
        ;
MEASUR: JSR      INCRA  ; RUN THRU THE INCREMENTS
        ; HAVE THE VALUE AT THIS POINT, DO WHAT THE APPLICATION
        ; REQUIRES--REMEMBER, TO CREATE REAL VALUE MUST MULTIPLY
        ; RESULT BY (VREF-VMAX)/TOTAL COUNT AND THEN SUBTRACT
        ; THAT RESULT FROM VMAX--DO IT IN DECIMAL OR BINARY, WHICHEVER
        ; IS BEST FOR THE APPLICATION
        LBI      1, 11  ; MAKE SURE SPACE IS CLEARED
        JSRP     CLEAR
        LBI      2, 11
        JSRP     CLEAR
        JSR      INCRB  ; FOR TEST-KEEP IT CLOSE
        LBI      1, 11  ; MAKE SURE COUNTER IS CLEARED
        JSRP     CLEAR
        JP       CLEAR2
INCRA:  LBI      1, 14
        STII     4      ; PRESET HERE FOR SMALLER COUNT
        STII     15     ; PRESET THE COUNTER FOR 4096
INCR1:  DGI      2      ; APPLY VIN AND ENABLE FEEDBACK
INCR:   LBI      1, 12
        SC
BINAD1: CLRA
        ASC
        NOP
        XIS
        JP       BINAD1
        NOP      ; 2 NOPS TO EQUALIZE TIMES
        NOP
        SKC
        JP       INCR
        DGI      0      ; DONE, NOW APPLY VREF
INCR2:  LBI      2, 12  ; COUNT UNTIL COMPARATOR CHANGES
        SC
BINAD2: CLRA
        ASC
        NOP
        XIS
        JP       BINAD2 ; STRAIGHT LINE THE ADD FOR SPEED
        ININ     ; SAVE WORDS BY USING G
        AISC     8      ; SEE IF IN3=1
        JP       INCR2 ; IN1 IS 0, KEEP COUNTING
OUTPUT: DGI      1      ; CLEAR THE CAPACITOR, APPLY VREF
        RET
INCRB:  LBI      1, 14  ; MAKE THE PASS FOR CAP INIT SHORT
        STII     7
        STII     15
        JP       INCR1

```

Figure 17A. Modified Dual Slope Code

## V. Voltage to Frequency Converters, VCO's

### A. BASIC APPROACH

The basic idea of this scheme is simply to use the COP420 to measure the frequency output of a voltage to frequency converter or VCO. This frequency is in direct relation to the input voltage by the very nature of such devices. There are really only two limiting factors involved. First of all, the maximum frequency that can be measured is defined in the microcontroller by the amount of time required to test an input and increment a counter of the proper length. With the COP420 this upper limit is typically 10 to 15kHz. The other limiting factor is simply the accuracy of the voltage to frequency converter or VCO. This accuracy will obviously affect the accuracy of the result.

Two basic implementations are possible and their code implementation is not significantly different. First, the number of pulses that occur within a given time period may be counted. This is straightforward and fairly simple to implement. The crucial factor is how long that given time period should be. To get the maximum accuracy from this implementation the time period should be one second. Such a time period would allow the distinction between the frequencies of 5000Hz and 5001Hz for example (assuming the V to F converter was that accurate or precise). Decreasing the amount of time will decrease the precision of the result. The alternate approach is to measure (by means of a counter) the amount of time between two successive pulses. This period measurement is only slightly more complicated than the pulse counting approach. The approach also makes it possible to do averaging of the measurement during conversion. This will smooth out any changes and add stability to the result. The time measurement technique is also faster than the pulse counting approach. Its accuracy is governed by how finely the time periods can be measured. The greater the count that can be achieved at the fastest input frequency — shortest period — the more accurate the result.

Figure 18 illustrates the basic concept. Figure 19 shows the flow charts and code implementation for both of the approaches discussed above. Note that whatever type of V to F converter is used, the code illustrated in Figure 19 is not significantly changed. In the code of Figure 19, the interrupt is being used to test an input and thereby decreases the total time loop.

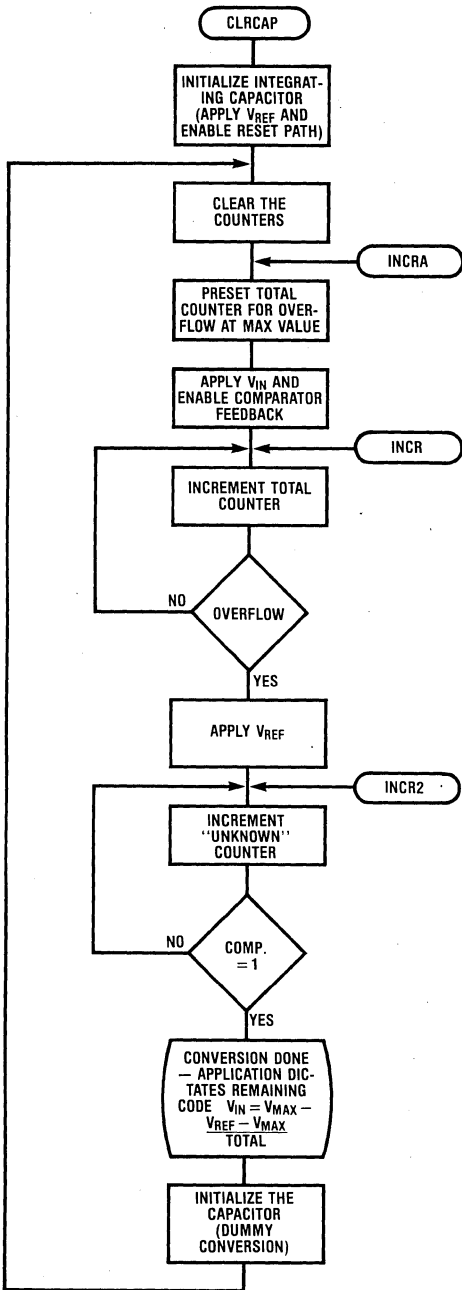


Figure 17B. Modified Dual Slope Flow Chart

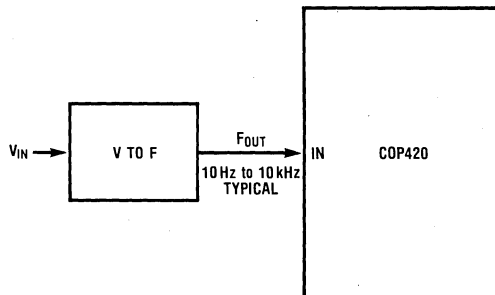


Figure 18. V to F Converter — Basic Concept





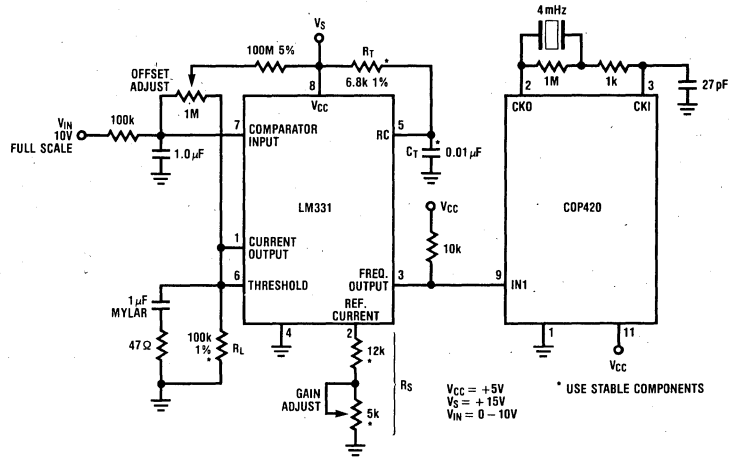


Figure 20. Basic LM331 Connection

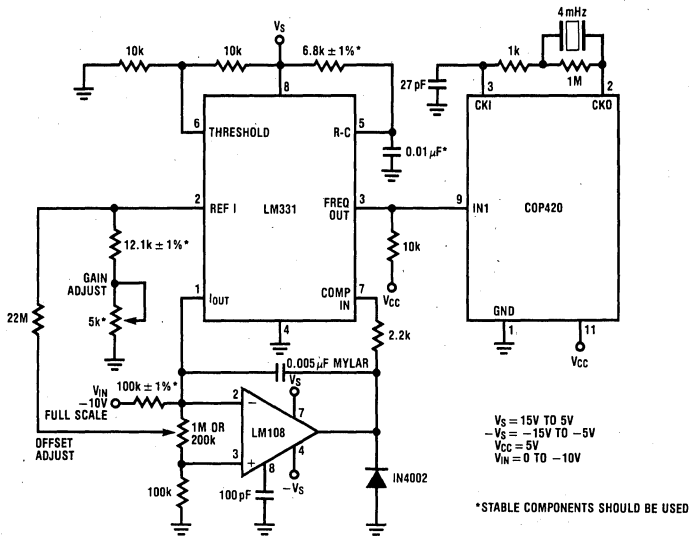


Figure 21. A to D with Precision Voltage to Frequency Converter

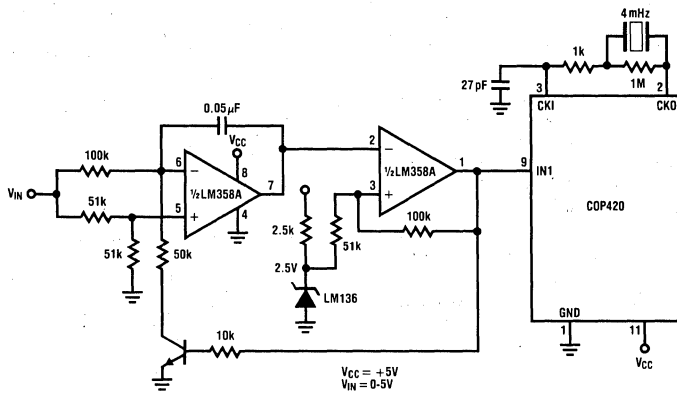


Figure 22. A to D with VCO

## VI. Successive Approximation

### A. BASIC APPROACH

The successive approximation technique is one of the more standard approaches in analog to digital conversion. It requires a counter or register (here provided by the COP420), a digital to analog converter, and a comparator. Figure 23 illustrates the basic idea with the COP420. In the most basic scheme, the counter is reset to zero and then incremented until the voltage from the digital to analog converter is equal to the input voltage. The equality is determined by means of the comparator. Figure 24 illustrates the flow chart and code for this most basic approach. The preferred approach is illustrated in Figure 25. This is the standard binary search method. The counter or register is set at the midpoint and the "delta" value set at one half the midpoint. The "delta" value is added or subtracted from the initial guess depending on the output of the comparator. The "delta" value is divided by 2 before the next increment or decrement. The method repeats until the desired resolution is achieved. While this approach is somewhat more complicated than the basic approach it has the advantage of always taking the same amount of time for the conversion regardless of the value of the

input voltage. The conversion time for the basic approach increases with the input voltage. The preferred approach is almost always faster than the basic approach. The basic approach is faster only for those voltages near zero where it has only a few increments to perform.

The accuracy of the approach is governed by the accuracy of the digital to analog converter and the comparator. Thus, the result can be as accurate as one desires depending on the choice of those components. Digital to analog converters of various accuracies are readily available as standard parts. Their cost is usually in direct relation to their accuracy. The reader should refer to the National Semiconductor Data Acquisition Handbook for some possible candidates for digital to analog converters. It is not the purpose here to compare those parts. The COPS™ interface to these parts is generally straightforward and follows the basic schematics shown in Figure 23. The user should take note and make sure the input and output ports of the converter are compatible — in terms of voltages and currents — with the COPS device. This is generally not a problem as most of the parts are TTL compatible on input and output. The precautions and restrictions as to the use of any given device are governed by that device and are indicated in the respective data sheets.

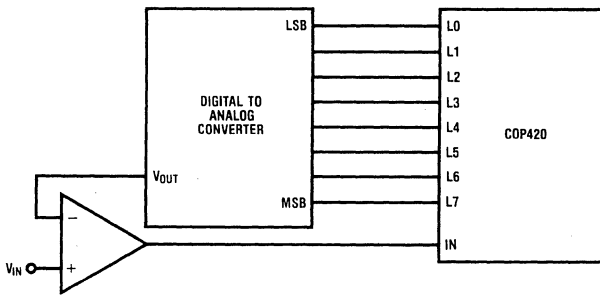


Figure 23A. Basic Parallel Implementation

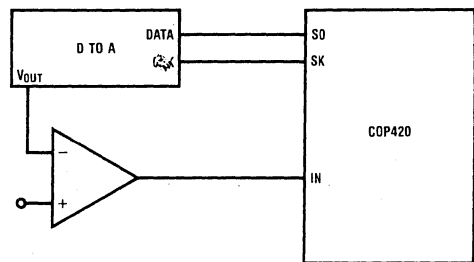


Figure 23B. Basic Serial Implementation

```

; 8 BIT SUCCESSIVE APPROXIMATION--BASIC SCHEME
; COMPARATOR INPUT TO COP = IN3
; OUTPUTS TO D TO A ARE L7 THRU L0 WITH L7 = MSB, L0 = LSB

CONVRT:  LBI    2, 14  ; SET THE RESULT VALUE TO ZERO
         STII   0
         STII   0
         LEI    4      ; ENABLE THE L PORT AS OUTPUTS
         JP     OUTPUT
INCR:    SC      ; ROUTINE FOR INCREMENTING THE RESULT VALUE
PIUSJ:   CLRA
         LBI    2, 14
         ASC
         NOP
         XIS
         JP     PLUS1
OUTPUT:  LBI    2, 15  ; SEND THE RESULT VALUE, STORED IN 2, 15-2, 14 TO
         LD     ; Q AND THEREBY OUT THROUGH L
         XDS
         CAMQ
         JSR    DELAY ; THIS IS ANY CONVENIENT ROUTINE TO MAKE SURE
                   ; THAT THE COP DOES NOT TEST THE COMPARATOR UNTIL
                   ; THE D TO A CONVERTER HAS HAD ENOUGH TIME TO DO
                   ; THE CONVERSION--THE AMOUNT OF TIME REQUIRED
                   ; IS CLEARLY DEPENDANT UPON THE D TO A CONVERTER
                   ; USED
         ININ   8      ; NOW READ THE COMPARATOR INPUT TO COP
         AISC   8      ; COULD SAVE A WORD IF USE G LINE AS INPUT
         JP     INCR  ; INPUT VOLTAGE STILL > CONVERTED ANALOG VOLTAGE

; CONVERSION DONE AT THIS POINT--THE COMPARATOR HAS CHANGED STATE
; HENCE, CONVERTED ANALOG VOLTAGE > INPUT VOLTAGE--SO STOP

```

Figure 24A. Code for Basic Approach of Successive Approximation

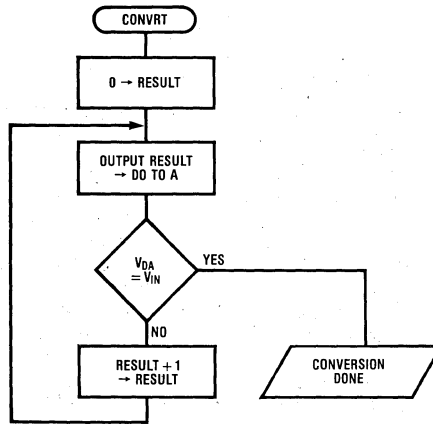


Figure 24B. Basic Approach, Successive Approximation

```

; 8 BIT BINARY SEARCH SUCCESSIVE APPROXIMATION
; INPUT TO COP IS IN3, L BUS IS OUTPUT TO D TO A, L7=MSB, L0=LSB
; COMPARETOR=0 WHEN D TO A VOLTAGE > VIN, OTHERWISE = 1

BINSRH: LBI 3, 14 ; SET INCREMENT = MAX VALUE/2 (WILL BECOME
          STII 0 ; MAX VALUE/4 BEFORE FIRST USE)
          STII 8
          LBI 2, 14 ; SET INITIAL VALUE OF RESULT TO MAX VALUE/2
          STII 0
          LEI 4 ; ENABLE THE L BUS AS OUTPUTS
          LBI 1, 15 ; NOW SET UP THE BIT COUNTER-OVERFLOW WHEN 8 BITS
          CLRA
          AISC 9 ; DO IT THIS WAY FOR COMPATIBILITY WITH INCREMENT
          OUTPUT: X 3 ; SAVE THE BIT COUNTER VALUE AND POINT TO RESULT
          LD
          XDS ; SEND THE RESULT TO G AND HENCE TO L
          CAMQ

DIVIDE: LBI 3, 15 ; DIVIDE THE INCREMENT VALUE BY 2, CAN BE DONE
          LD ; IN SEVERAL WAYS SINCE THIS IS A VERY SPECIAL
          AISC 8 ; PURPOSE DIVIDE FUNCTION
          JP DIV1 ; ALSO, DO THE DIVIDE HERE TO GIVE THE D TO A TIME
          STII 4 ; TO DO THE DIGITAL TO ANALOG CONVERSION

DIV1: AISC 4
       JP DIV2
       STII 2

DIV2: AISC 2
       JP DIV3
       STII 1

DIV3: LBI 3, 14
       AISC 1
       JP DIVA
       STII 8
       STII 0
       ; DEPENDING ON THE D TO A USED, MAY NEED MORE DELAY HERE
       ; MUST BE SURE THE RESULT IS STEADY BEFORE TEST THE COMPARETOR

TEST: LBI 3, 14
      ININ
      AISC 8 ; COULD SAVE A WORD IF USED G LINE AS INPUT
      JP INCR

DECR: SC ; INPUT LESS THAN D TO A CONVERTED VOLTAGE
      SUB: 1 ; SUBTRACT THE INCREMENT VALUE FROM RESULT
      CASC
      NOP
      XIS 1
      JP SUB
      BITPL1

INCR: RC ; INPUT > D TO A CONVERTED VOLTAGE
      LD 1 ; ADD THE INCREMENT VALUE TO RESULT VALUE
      ASC
      NOP
      XIS 1
      JP ADD

BITPL1: LBI 1, 15 ; NOW INCREMENT BIT COUNTER TO SEE IF DONE
         LD
         AISC 1
         JP OUTPUT
         ; CONVERSION DONE AT THIS POINT
  
```

Figure 25A. Binary Search Successive Approximation Code

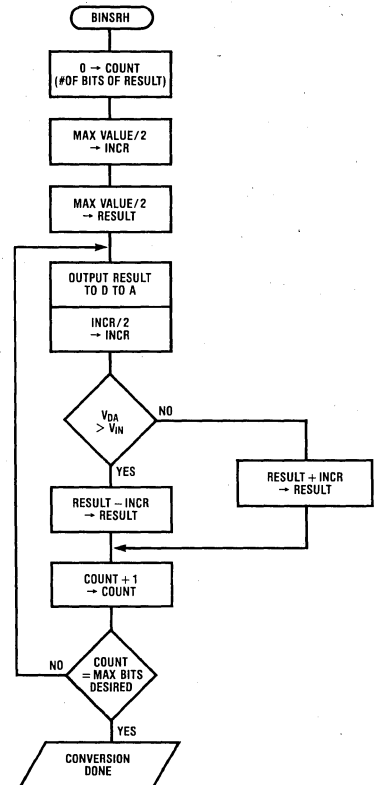


Figure 25B. Binary Search Successive Approximation Flow Chart

## B. SOME COMMENTS ON RESISTOR LADDERS

If the user does not wish to use one of the standard digital to analog converters, he can always build one of his own. One of the most standard methods of doing so is to use a resistor ladder network of some form. Figure 26 illustrates the basic forms of binary ladders for digital to analog converters. The figures also show the transition from the basic binary weighted ladder in Figure 26A to the standard R-2R ladder Figure 26C.

Consider Figure 26A. The choice of the terminating resistor is made by hypothesizing that the ladder were to go on ad infinitum. It can then be shown that the equivalent resistance at point X in that figure would be equal to  $128R$ , the same value as the resistor to the least significant bit output. This fact is used to create the intermediate ladder of Figure 26B. This step is done because it is usually undesirable to have to find the multitude of resistor values required in the basic binary ladder. Thus, the modification in Figure 26B significantly reduces the number of resistor values required. As stated earlier, the resistance looking down the ladder at point X in Figure 2 is equal to the resistor connected to the binary output at that point; here the value is  $2R$ . Remembering the objective is to minimize the number of different values required, if we simply use the same R-2R arrangement as before with a termination of  $2R$  we get an effective resistance at point Y of Figure 26B or  $0.5R$ . This means that a serial resistance of  $1.5R$  is required to maintain the integrity of the ladder. If we carry this on through 8 bits, the circuit of

Figure 26B results. From this it is only a small step to create the standard R-2R network. The analysis is the same as done previously.

There is absolutely no restriction that the ladders must be binary. A ladder for any type of code can be constructed with the same techniques. Ladders comparable to Figures 26A and 26B are shown in Figure 27 for a standard 8421 BCD code. With the BCD code, the input must be considered in groups of digits with four bits creating one digit. This is the direct analog of 1 binary digit per input. We need four inputs to create one decimal digit. Thus the resistor values in each decimal digit are 10 times the values in the previous decimal digit just as the resistor value for each successive binary digit was twice the value for the preceding binary digit. Note that this analysis can be easily extended to any code. The termination resistance is calculated in the same manner — assume the decimal digit groupings extend out to infinity. It can be shown that the resistance of the ladder at point X in Figure 27A is  $480R$ . Thus Figure 27A represents the basic 8421 BCD ladder for three digit BCD number. This termination resistance will vary with where it is placed. Basically this resistance is equal to nine times (for a decimal ladder) the parallel resistance of the last digit implemented. (This relation can be shown mathematically if one desires, the multiplier is a function of the type of ladder used — multiplier = 1 for binary systems, 9 for decimal systems, etc.) Thus the termination resistance would be  $48R$  if the network were terminated after the 2nd digit and  $4.8R$  if the network were terminated after the 1st digit implemented. In Figure 27B

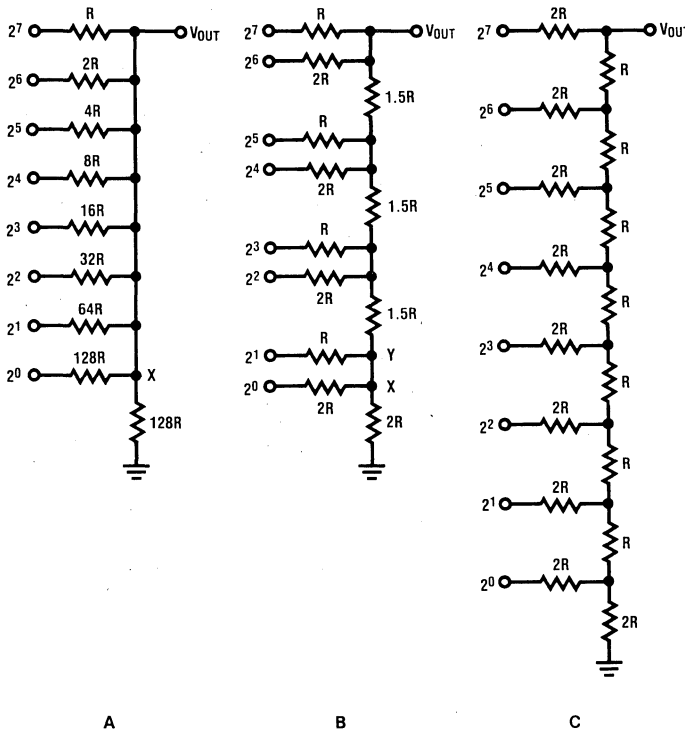


Figure 26. Binary Ladders



we are attempting to use only the resistor values for one decimal digit. This means that the last terminating resistor must be a  $4.8R$  by the analysis above. Thus at point X in Figure 27B we must have an equivalent of resistance of  $4.8R$ . The equivalent resistance at point Y of Figure 27B, looking down from the ladder, is  $0.48R$ . Thus the other series resistance must be  $4.32R$  ( $4.8R - 0.48R$ ). Thus the network of Figure 27B results.

Generally, ladders can be very effective tools when understood and used properly. They can be significantly more involved than indicated here. There are a number of texts and articles that cover the subject very nicely and the reader is referred to them if more information on ladder design, the use of ladders, and advanced techniques with ladders is desired.

One final note is of some interest. The ladders may be readily constructed for any type of code to create the analog voltage. Note that there is no restriction that the code, or the ladder network, be linear. Thus, effective use of ladder networks may significantly reduce system difficulties and complexities caused by the fact that the

analog to digital conversion is being performed on a voltage source that changes nonlinearly, for example a thermistor temperature probe. By using the properly designed ladder network, the nonlinearity can effectively be eliminated from consideration in the code implementation of the analog to digital conversion.

The accuracy of ladders is a direct function of the accuracy of the resistors and the accuracy of the voltage source inputs. This is obvious since the analog voltage is in fact created by means of equivalent voltage dividers created when the various inputs are on or off. It is also essential that the ladder sources be the precise same value at all inputs to the ladder network. If this is not the case, errors will be introduced. In addition, the output impedance of the voltage source should be as small as possible. The success of the ladder scheme depends on the ratios of the resistance values. Inaccuracies are introduced if those ratios are disturbed. Some possible implementations of the successive approximation approach with a ladder network used for the digital to analog conversion are

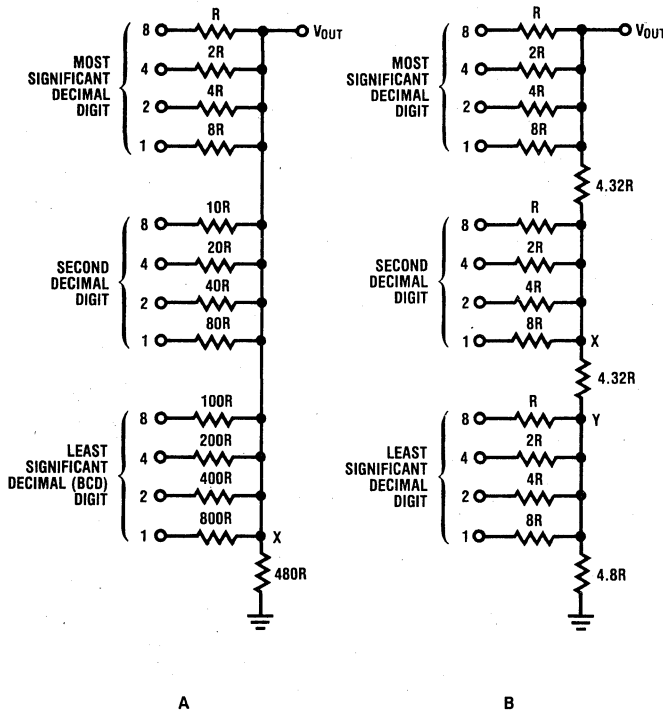


Figure 27. 8421 BCD Ladders

indicated in Figure 28. Note that these are functional diagrams. Feedback or hysteresis for comparator stabilization are not shown. The reader should be aware that his particular application may require that these factors be considered. Figure 28A is the simplest scheme and also the least accurate. With little or no load, the high output level of the L buffer should be very close to  $V_{CC}$  and the low level close to ground. Also the output impedance of the buffers must be considered. Therefore, rather large resistor values are used — both to keep the load very small and to dwarf the effect of the

output impedance. With the configuration in Figure 28A, four bit accuracy is about the best that can be achieved. By being extremely careful and using measured values, an additional bit of accuracy may be obtained but care must be used. However, the schematic of Figure 28A is very simple. Figure 28B represents the next step of improvement. Here we have placed CMOS buffers in the network. This eliminates the output impedance and reduces the level problems of the circuit of Figure 28A. The CMOS buffer will swing rail to rail, or nearly so. The accuracy of  $V_{CC}$  and the

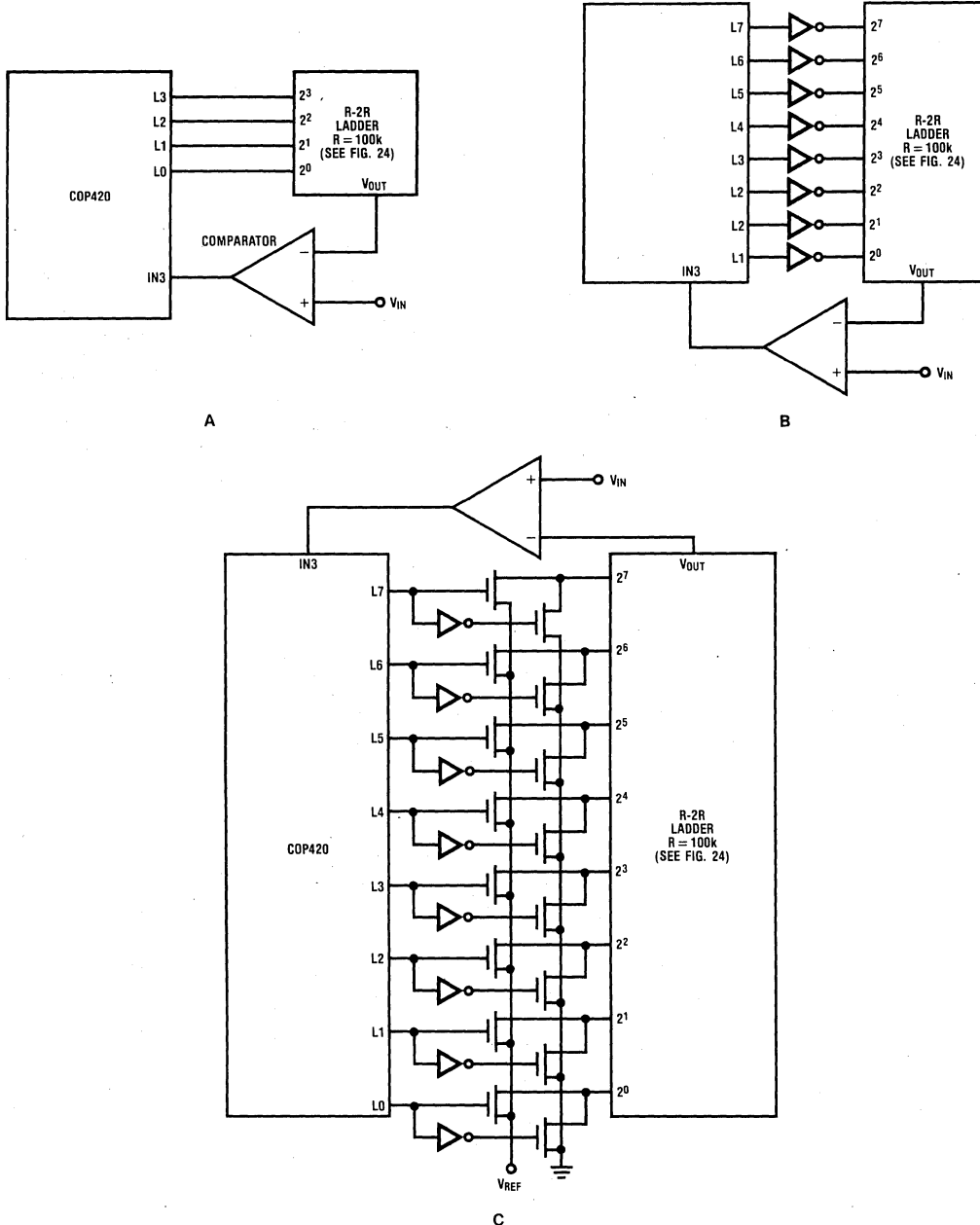


Figure 28. Interfaces to Ladder Networks



Figure 30 is the flow chart and code required to do the interfacing. As can be seen, the overhead in the COP420 device is very small. The choice of inputs and outputs is arbitrary. The only pin that is more or less restricted is the use of SK as the clock for the converter. SK is clearly the output to use for that function as, when properly enabled, it provides pulses at the instruction cycle rate.

### C. NAKED-8™ INTERFACE

The Naked-8 family of analog to digital converters (ADC0801, ADC0802, ADC0803, ADC0804) is very easy to

interface and is generally a very useful offboard converter. The interface is not significantly different from that of the ADC0800, but the Naked-8 is a much better device. The four control signals are somewhat different, although there are still four control lines. Here we have a chip select, a read, a write, and an interrupt signal. All are negative going signals. Start conversion is the anding of chip select and write. Output enable is the anding of chip select and read. The interrupt output is an end convert signal of sorts. The device may be clocked externally or an RC may be connected to it and it will generate its own clock for the conversion. In addition the device has differential inputs which allow

```

MEASUR: LEI    0      ; FLOAT THE L LINES
        SC
START12: CLRA          ; MAKE SURE SD STAYS ZERO
        XAS          ; MAKE SURE SK STAYS CLOCK
        OGI    2      ; SEND START PULSE
        OGI    0
        LBI    2, 13
READ11: ININ
        AISC    14     ; WAIT FOR EOC SIGNAL
        JP     READ11
        OGI    4      ; HAVE EOC, ENABLE OUTPUTS
        INL          ; READ THE L LINES
        X
        COMP          ; CREATE PROPER POLARITY
        XDS
        COMP
        X
        OGI    0      ; DISABLE ADC0800 OUTPUT
        ; HAVE THE RESULT AT THIS POINT--USE IT IN WHATEVER
        ; MANNER IS REQUIRED BY THE APPLICATION
        LBI    2, 10
        JSRP   CLRR
        JP     MEASUR
  
```

Figure 30A. A to D with ADC0800

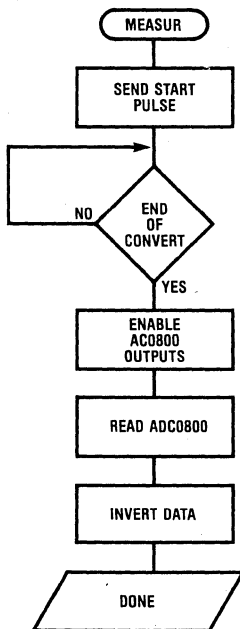


Figure 30B. ADC0800 Interface Flow

the 8-bit conversion to be performed over a given window or range of input voltages. The reader should refer to the Naked-8™ data sheet for more information. Figure 31 indicates a basic interface of the Naked-8 to

the COP420. Again, the interface is simple and straightforward. The code required to interface to the device is minimal. Figure 32 illustrates the flow chart and code required to do the interface.

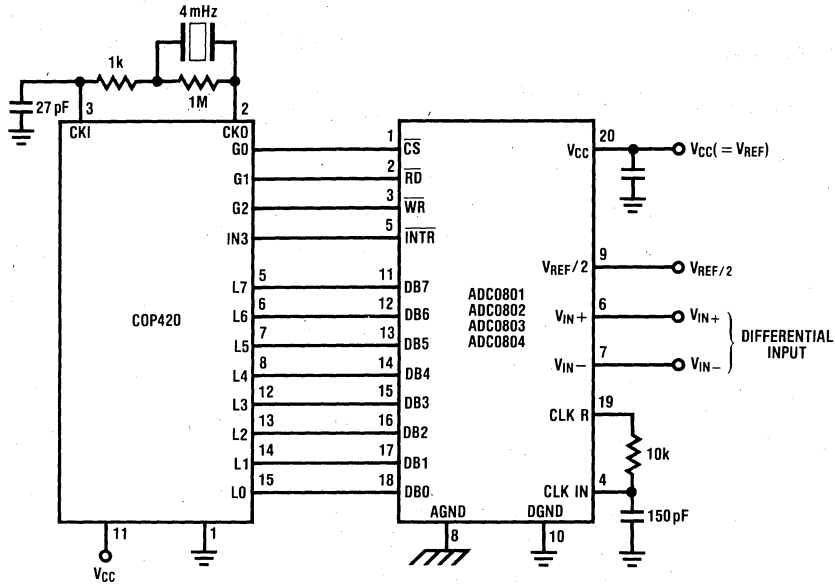


Figure 31. COP420 — Naked-8 Interface

```

; INTERFACE TO NAKED 8(TM)
;
NAKED8:  OGI    15    ; SET ALL G LINES HIGH (USUALLY DONE AT
; POWER UP
LOOP?:   LEI    0     ; TRI STATE THE L LINES FOR READING
; SEND CHIP SELECT LOW (CS BRACKETS OTHER SIGNAL)
OGI     14
OGI     10    ; CS LOW AND WR LOW = START CONVERSION
OGI     14    ; RAISE WR
OGI     15    ; RAISE CS, NAKED 8 IS NOW CONVERTING
LOOP?:   ININ   8     ; WAIT FOR THE INTR SIGNAL--COULD SAVE THIS TEST
; IF USED IN1 AND THE INTERRUPT FEATURE OF COP4
AISC    8
JP      READ  ; INTR IS LOW, DATA IS READY
JP      LOOP2
; SET UP RAM LOCATION FOR READ
RI-AD:  LBI    0,0
OGI     14    ; SEND CS
OGI     12    ; SEND CS AND READ = OUTPUT ENABLE
NOP     ; WAIT--NEED WAIT ONLY 125NS, BUT 1 CYCLE IS MIN
; TIME WE CAN WAIT
INL     ; READ THE L LINES
OGI     15    ; TURN OFF THE NAKED 8--CS AND RD HIGH
;
; DONE AT THIS POINT, DO WHATEVER IS REQUIRED WITH THE RESULT
;

```

Figure 32A. COP420/Naked-8 Sample Interface Code

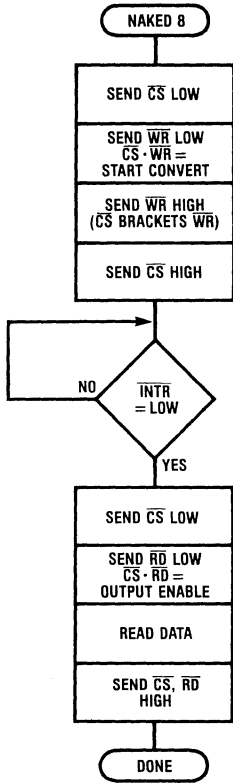


Figure 32B. COP420/Naked-8 Interface Flow

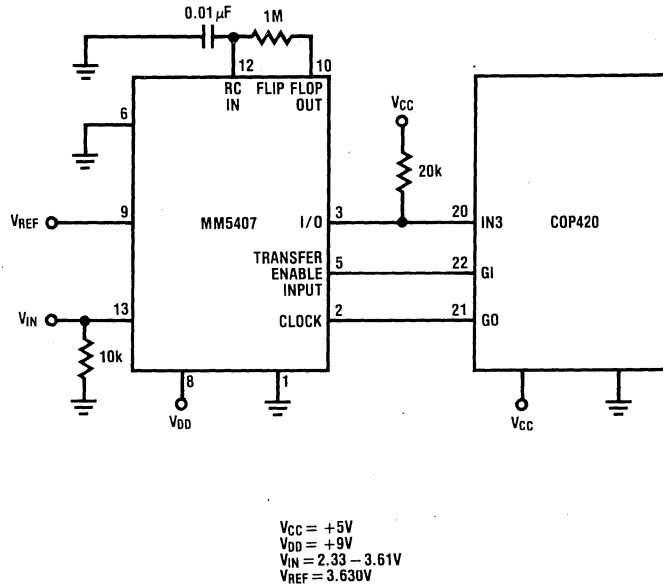


Figure 33. MM5407 Interface

#### D. THE MM5407 AS AN A/D CONVERTER

The MM5407 is a digital thermometer usually used in conjunction with the MM5406 digital clock. However, the MM5407 can make a very effective analog to digital converter. The heart of the MM5407 is, in fact, an analog to digital converter. The device is designed to interface directly with the LM134 temperature transducer which produces an output voltage related to temperature. The relationship is 10mV per degree Kelvin. The MM5407 is specified to operate from  $-40^{\circ}C$  to  $+88^{\circ}C$  ( $233^{\circ}K$  to  $361^{\circ}K$ ). The device provides a serial output with the result in either centigrade or fahrenheit. The accuracy is  $\pm 2^{\circ}F$ .

Now, translating all of this into the pertinent information that we need we get the following: The MM5407 will perform an analog to digital conversion for input voltages in the range of 2.33 volts to 3.61 volts. The result is accurate to about  $\pm 10$  millivolts. This translates to an accuracy of 7 bits  $\pm 1$  bit. The interface,

as shown in Figure 33 is not complex. Note that here SK is not being used for the clock because SK is too fast. The clock input on the MM5407 has an upper limit of 10kHz. Also because of the speed, we are using IN3 rather than serial in as the input from the MM5407. Note also that the MM5407 is a nine volt device although the interface signals are TTL compatible. The COP420 is a 5 volt device. However, the COP420L will run at 9 volts and thereby remove a requirement for two power supplies. If the user system has dual supplies, the dual supply requirement is not serious.

Once the data is read into the COP<sup>SM</sup> device, the processing required is simple. One need only add 273 to the number received (if the MM5407 is operated in the Centigrade mode) to create the proper voltage value. Obviously, if a different range is desired, it would be possible to do some scaling at the input of the MM5407 to create the proper voltage. The COPS device would then have to account for this scaling — generally, a straightforward task.

```

; CODE FOR MM5407/COP420 AS A TO D CONVERTER
; GO AND G1 ARE HIGH ON ENTRY TO THE ROUTINE

MM5407: CLRA          ; RUN A FEW CLOCKS TO DO THE CONVERSION
        AISC         8
        LBI         2, 12
LOOP:   X
        JSRP        CLOCK2
        NOP
        LD
        AISC         1
        JP          LOOP
        STII        0      ; NOW CLEAR OUT THE MEMORY FOR READING
        STII        0
        STII        0
        STII        0      ; 0 TO 2, 12 THRU 2, 15
START:  LBI         2, 12  ; NOW SEND START TRANSMIT SIGNAL AND MAINTAIN
        JSRP        CLOCK1 ; TIMING
        NOP
        JSRP        CLOCK2
        NOP
        JSRP        CLOCK2
        NOP
        JSRP        CLOCK2
        NOP
READ:   JSRP        CLOCK2 ; NOW READY TO READ THE DATA(16 BITS)
        SMB         3      ; ALLOW FOR THE COMPLEMENT DATA ON THE READ
        JSRP        CLOCK2 ; I. E., COMPLEMENT THE INFO. WHEN READING IT
        SMB         2
        JSRP        CLOCK2
        SMB         1
        JSRP        CLOCK2
        SMB         0
        LD          ; NOW TEST TO SEE IF DONE
        XIS
        JP          READ  ; NOT YET FINISHED
        LBI         2, 13  ; NOW JUGGLE THE DATA TO PUT IT IN MORE DESIRAB
        CLRA        ; FORM--MINUS/BLANK, TENS, UNIT
        X           ; IGNORE 2, 12 BECAUSE WE KNOW IS CENTIGRADE MODE
        LBI         2, 15  ; REFER TO MM5407 DATA SHEET
        X           ; INFO WAS IN FORM: UNITS, TENS, MINUS/BLANK
        LBI         2, 13
        X
        LBI         2, 15  ; NOW TEST TO SEE IF IS MINUS
        X           ; ACCUMULATOR IS ZERO PRIOR TO THIS EXCHANGE
        AISC         5      ; TEST FOR THE MINUS CODE
        JP          ADD273
COMPL:  SC          ; IS MINUS, TAKE TENS COMPLEMENT OF NUMBER
        LBI         2, 13  ; ALSO, ZERO IS IN MINUS POSITION
COMP2:  CLRA
        X
        CASC
        ADT
        XIS
        JP          COMP2
ADD273: LBI         1, 13  ; NOW SET UP TO ADD 273 TO THE RESULT
        STII        3
        STII        7
        STII        2

```

Figure 34A. MM5407/COP420 A/D Interface Code

```

RC
LBI 1, I3
ADDLP: LD 3
      AISC 6
      ASC
      ADT
      X15 3
      JP  ADDLP
; FINISHED AT THIS POINT, DO ANY REQUIRED SCALING, ETC. HERE
RET
; PAGE 2 ; THE REQUIRED SUBROUTINES HERE
CLOCK: CLRA
      OGI 0 ; SEND CLOCK AND START SIGNAL LOW
      JP  CLK
CLOCK?: OGI 2 ; SEND CLOCK ONLY LOW
      CLRA
CLK: AISC 3 ; MAKING SIMPLE TIMING LOOP--HERE ADJUSTING FOR
      JP  -1 ; TOTAL PERIOD = 100us(25 CYCLE TIMES AT 4us
      AISC 4 ; INSTRUCTION CYCLE TIME)--HERE USING 13 CYCLE
      JP  -1 ; TIMES ON, 12 CYCLE TIMES OFF
      OGI 3 ; SET CLOCK BACK HIGH
      NOP 3 ; THESE NOP'S FOR TIMING ONLY
      NOP
      NOP
      NOP
      ININ ; READ THE INPUT LINE(I3)
      AISC 8
      RET
      RETSK

```

Figure 34A. MM5407/COP420 A/D Interface Code, cont'd

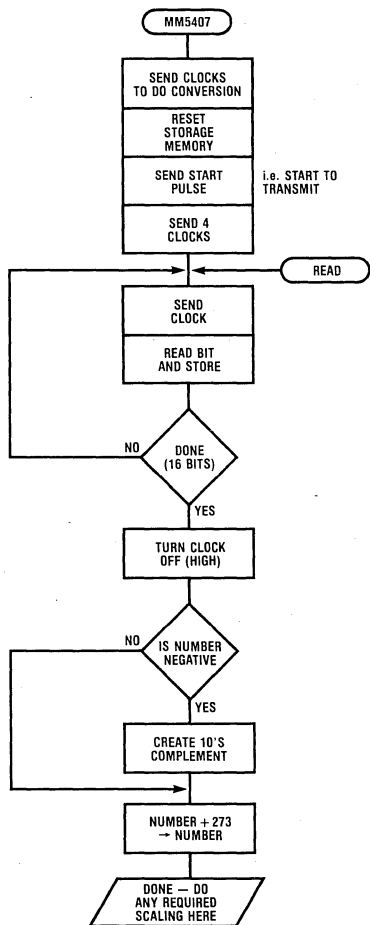


Figure 34B. MM5407 as A/D Converter Flow Chart



## VIII. Conclusion

Several analog to digital techniques using the COPS™ family have been presented. These are by no means the only techniques possible. The user is limited only by his imagination and whatever parts he can find. The COPS family of parts is extremely versatile and can readily be used to perform the analog to digital conversion in almost any method. Generally, those techniques where the COPS device is doing the counting or timekeeping are slow. However, those techniques are generally slow inherently. The fastest methods are those where the conversion is being done offboard and the COPS device is merely reading the result of the conversion when required. Also, an attempt has been made to illustrate the lower cost techniques of analog to digital conversion. This, by itself, restricts most of the techniques described to about 8-bits accuracy. As was mentioned several times, the greater the accuracy that is desired the more accurate the external circuits must be. Ten and twelve-bit accuracies, and more, require references that are accurate. These get very expensive very rapidly. There is nothing inherent in the COPS devices that prevents them from being used in accurate systems. The precautions are to be taken in the system regardless of the microcontroller. The only problem is that, in those accurate systems where the COPS device is doing the timekeeping and counting, this increased accuracy is paid for by increased time to perform the conversion.

Several devices have been used in conjunction with the COPS device in the previous sections. It is again recommended that the user refer to the specific data sheets of those devices when using any of those circuits. It must again be mentioned that the standard precautions when dealing with analog signals and

circuits must be taken. These are described in the National Semiconductor Linear Applications Handbook and in the data sheets for the various linear devices. These precautions are especially significant when greater accuracy is desired.

The COPS family of microcontrollers has shown itself to be very versatile and powerful when used to perform analog to digital conversions. Most techniques are code efficient and the microcontroller itself is almost never the limiting factor. It is hoped that this document will provide some guidance when it is necessary to perform analog to digital conversion in a COPS system.

## IX. References

1. "Digital Voltmeters and the MM5330", National Semiconductor Application Note AN-155.
2. Walker, Monty, "Exploit Ladder Network Design Potential." Part One of two part article on ladder networks. Magazine and date unknown.
3. Wyland, David C., "VFC's give your ADC design high resolution and wide range." *EDN*, Feb. 5, 1978.
4. Redfern, Thomas P., "Pulse Modulation A/D Converter" *Society of Automotive Engineers Congress and Exposition Technical paper #780435*, March 1978.
5. National Semiconductor Linear Applications Handbook, 1978.
6. National Semiconductor Linear Databook, 1980.
7. National Semiconductor Data Acquisition Handbook, 1978.



## Introduction

As part of National Semiconductor's continuing effort to define and implement a full spectrum of COPS Television Controllers (CTCs), this document will describe progress made in programming a COP420 to serve as a prototype 'low-end' CTC. Used in conjunction with an MM5439 Phase Locked Loop (PLL) and an MM5450 display driver, this processor allows a television receiver to have the following functions:

1. Frequency Synthesis Tuning
2. Keyboard Scan and Decode
3. MM53126 Format Serial Decode
4. 64 Level Analog Outputs
5. Direct Channel Entry
6. Channel and Fine Tune Slewing
7. Analog Output Slewing
8. LED Channel Display
9. Last Channel Memory

## System Overview

Shown in Figure 1, the heart of the CTC prototype hardware is the COP420 itself. This particular member of National's COPS family of 4-bit microcontrollers has 1024 bytes of program memory, 64 digits of scratch-pad RAM, 24 input and output pins, and an efficient 49-member instruction set. It is the workhorse of the television tuning

system and provides the processing power to scan the keyboard, decode the serial input, run the channel display, and control the PLL. System capabilities may be enhanced or scaled-down for different markets simply by changing the processor's algorithms. This flexibility combined with low-cost makes the COPS family, and in particular the COP420, a standout in the field of high-volume, low-to-medium range television controllers.

The MM5439 PLL is of next importance in the prototype system. Originally designed for the European Microprocessor Television Controller (MTC) market, the 5439 offers capabilities found in traditional PLL circuits as well as general purpose input and output pins and 6 pulse-width modulation D/A converters. This allows the COP420 to use it to band-switch the UHF and VHF tuners in addition to providing analog outputs for controlling television parameters such as volume, brightness, and color. The MM5439 operates with a 14-bit code and is capable of resolving the RF spectrum into 64 kHz steps; more than adequate for U.S. Television receivers.

The serial input of Figure 1 is generated by using an MM53126 infrared remote control circuit. The MM53126 scans and decodes a key closure and provides serial data to drive infrared transmitter diodes. At the receiving end, the infrared signal must be detected and amplified to provide a digital signal for the COP420. The COPS device provides the intelligence to receive the serial data

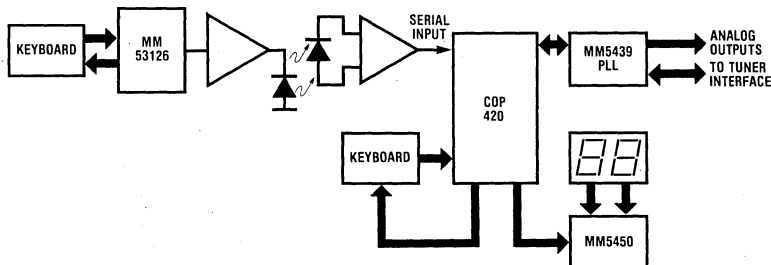


Figure 1. CTC Block Diagram

and to route program control just as if the key entry originated from the main keyboard.

The third circuit shown in Figure 1 is the MM5450 display driver. The 5450 is a direct drive, serial input, 35-segment LED driver. Due to its serial nature, it is best interfaced to the COPS' serial output port. The 5450 is gaining popularity because of its low-cost, adjustable high-current outputs, and low-noise non-multiplexed display format. Its sole duty in the system is to display the current channel number.

### Hardware Description

Utilizing the MM5439 as the system PLL dictated the basic structure of much of the prototype circuitry. The MTC series of components were designed to be MICROBUS™ compatible. That is, they were designed to connect to an 8-bit bi-directional data bus, address lines, and control strobes. The COPS™ family of processors does not possess a traditional bus structure, and to interface to a parallel bus device such as an MM5439 requires that COPS inputs and outputs emulate the data, address, and control bus functions. Figure 2 illustrates the use of the COPS L pins as the data bus, the G port for addressing, SK as a read strobe, SO as a write strobe, and DO as chip select.

Figure 2 also details the 5439 D/A, band-switching, and oscillator circuitry. The D/A interface is a simple capaci-

tor integrator that requires a current source from within the receiver chassis. UHF/VHF band-switching is accomplished by using 3 general purpose open-collector outputs to drive dual transistor 24 volt buffers. The one transistor 4.0MHz crystal oscillator also shown provides the stable reference needed by the PLL. In addition, it is used to generate a 4-microsecond instruction cycle within the COP420. This speed is necessary to insure that pulse-position-modulated (PPM) signals coming from the MM53126 are properly decoded.

The MM5439 and UHF/VHF tuner interface shown in Figure 2 is somewhat more complicated. By comparing the UHF/VHF local oscillator to the 4 MHz system clock, the 5439 generates two negative-going signals that are designed to raise or lower the varactor tuning voltage, and thus close the frequency synthesis loop. To accomplish this an LF351 is configured as a differential integrator to generate the tuning voltage. The single-pole filter on the output is to minimize transients. The PLL NMOS circuitry in the 5439 is not fast enough to handle the tuner local oscillator directly, so two counters are used to divide this frequency down. The SDA2001 ECL pre-scaler divides the frequency first by 64, and then the 74LS169 alternately divides by 15 or 16 under 5439 control.

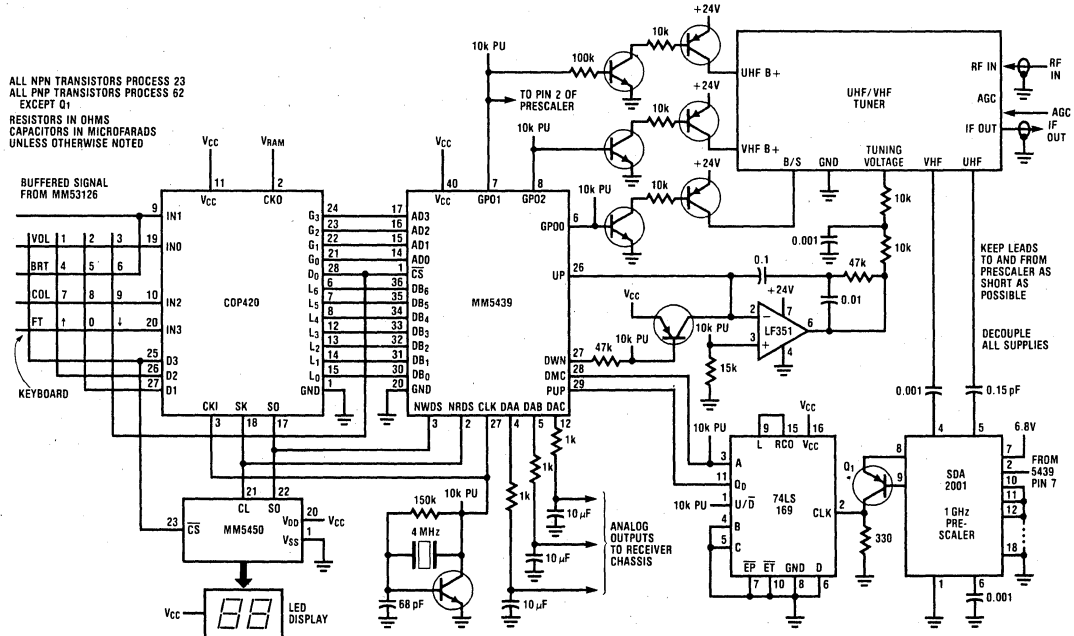


Figure 2. Low-End CTC Schematic

## Software Description

The major features of the software written for this low-end CTC implementation are described in the flowchart of Figure 3. Readily observable items of interest are the initialization, serial-input, delay, and instruction decode portions of the program. The function blocks comprising the PLL code calculations, serial processing, and display routines are less noticeable, but worthy of additional mention. They will now be summarized.

To successfully tune the television receiver a 14-bit code must be presented to the MM5439 PLL. This 14-bit

binary code is calculated from current BCD channel number using the following equation:

$$\text{PLL CODE} = \text{CHANNEL NUMBER} \cdot 6 \text{ MHz} + \text{BIAS}$$

The variable marked BIAS is necessary because there are gaps between channel groups in the American television RF spectrum. BIAS will have different values for the channel ranges 2-4, 5-6, 7-13, and 14-83.

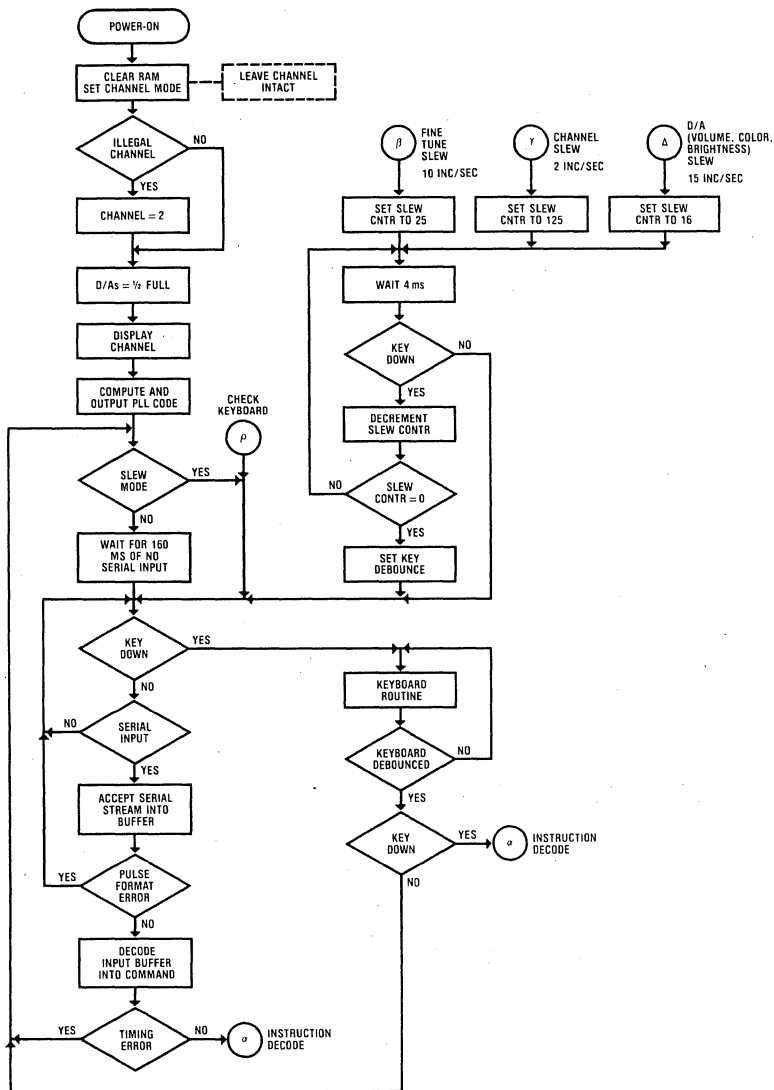


Figure 3. CTC Major Program Flow

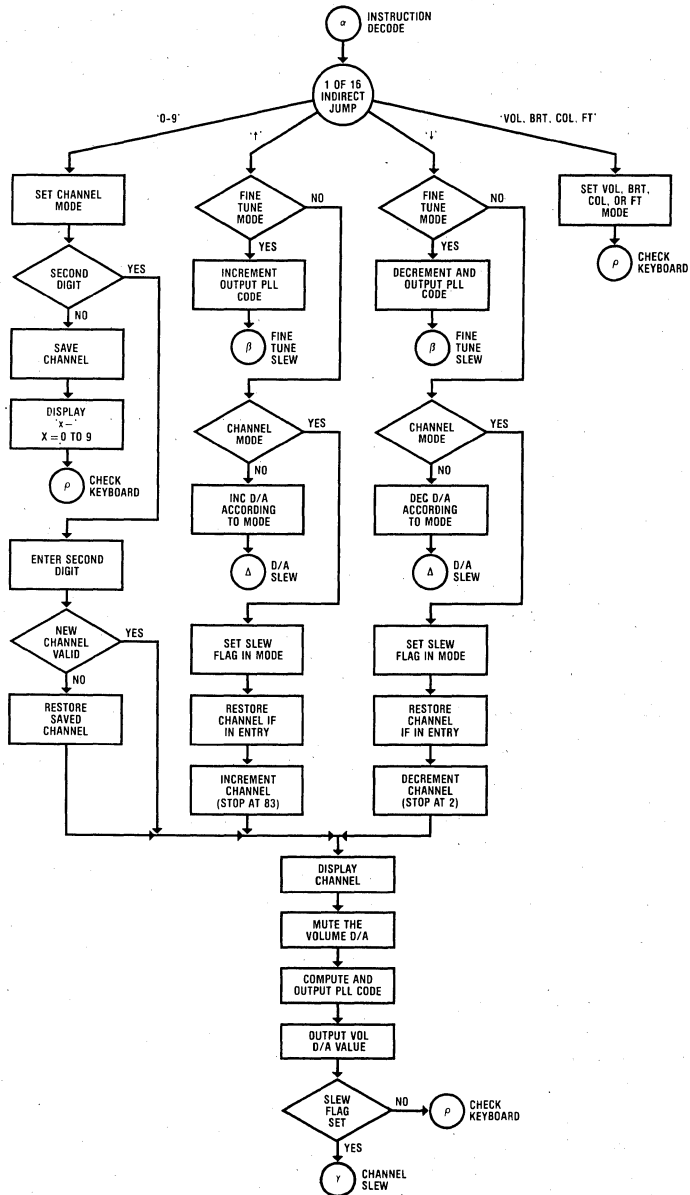


Figure 3. CTC Major Program Flow (cont'd)

The most time critical software operation encountered was processing the remote serial input stream. Speed considerations necessitated that this routine be broken into two portions, reading and decoding. Reading the stream required that the time between each pulse in the 14-bit code (counting start and stop bits) be saved in a unique memory location. Figures 4 and 5 illustrate the pulse timing and serial format. Only after all 14 bits were received could the timing be analyzed for validity and converted into a parallel code. Because the MM53126 generates a continuous stream of pulse packages during key depression, a form of debouncing was also needed on the input so only the first packet was decoded as an instruction.

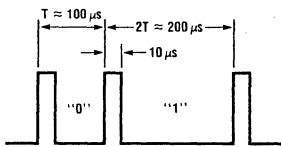


Figure 4. Pulse-Position-Modulation (PPM) Timing

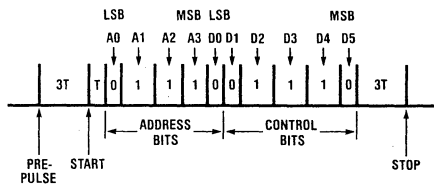


Figure 5. Format of Remote Control Signal 0111001110

The keyboard routine scans the key contacts by sweeping a logic low through the column outputs and checking for a resulting low on the row inputs. Once a key closure is sensed, it is converted into a unique 1 of 16 code and acted upon. It must then be released 64 milliseconds before a new key may be processed.

The last major routine shown only as a function block in the flowchart is the MM5450 display interface routine. In preparation to passing segment data to the 5450, the COP420 must first convert each digit of the channel number into its seven-segment display equivalent and place that information in a buffer. The final part of the display routine is simply serializing that buffer along with a start bit to the MM5450.

As previously stated, the COP420 has 64 digits of scratchpad RAM. Well designed data structures within this RAM will optimize overall program efficiency. With this in mind, the CTC structures were defined and assigned to

particular positions in memory. Table 1 breaks down the program data structures and lists the number of 4-bit digits needed for each. RAM efficiency for this program was 39/64 or approximately 60 percent.

Table 1. CTC RAM Allocation

Data Description	Digits Used
PLL Code and band data	5
Display and PLL word area	5
Remote input buffer	13
Remote command buffer	3
D/A mirror values	6
Current channel	2
Channel storage	2
Flags	2
Key decoding	2
Misc.	2
<b>Total</b>	<b>39</b>

Listed in Table 2 are the major routines in the low-end CTC program and their respective ROM usage. ROM efficiency in this case would be 780/1024 or 76 percent.

Table 2. CTC ROM Allocation

Routine Description	Bytes Used
Initialization	50
PLL code calculation	80
Increment, decrement, PLL I/O	130
Remote input	80
Remote input decoder	20
Keyboard	100
MM5450 display	50
7-segment look-up table	10
Channel check	20
Slew control	40
PLL fine tune	20
Instruction decoding and main loop	180
<b>Total</b>	<b>780</b>

## Conclusions

A COP420 has been shown to be ideal in performing the functions of a low-end television controller. Manufacturers integrating COPS devices into their television receiver designs would benefit from cost and capability advantages. Due to the fact that ROM and RAM are under utilized in the software described, it would be logical and cost-effective from a product viewpoint to expand the low-end concept and take full advantage of the COP420 by incorporating mid-range features into the controller software. Conversely, a lesser member of the COPS family could perform a subset of the functions presented in more cost-driven applications.

# Design Considerations for a COP420C-Based Telephone-Line Powered Repertory Dialer

National Semiconductor  
COP Note 3  
Chris Stacey  
August 1980



## Introduction

The COP420C is a CMOS member of the COPS™ low-cost microcontroller family. Its port flexibility and low power consumption make it an ideal controller for various functions which are attractive for a "smart" telephone set, capable of being powered from the telephone line.

Figure 1 illustrates a repertory dialer phone with a library of fifteen frequently used numbers, (plus the last number dialed) stored in a standard CMOS RAM. A push-button keyboard, scanned directly by the COP420C, enables telephone numbers to be keyed in and dialed out directly or a telephone number to be stored in the RAM and dialed automatically. An abbreviated code can be used for store access or separate keys provided for individual stored numbers. Either series or shunt mode loop-disconnect signalling can be generated with software routines under control of the internal timer. Alternatively, the system can control a dual tone multi-frequency generator. An expandable LCD display is also provided, using drivers interfaced via the COPS serial port. Features such as a real time clock, call timing, alarm and calculator can readily be added, either in software or with low-cost peripheral devices.

The circuitry shown can operate on a single supply rail as low as 3V, consuming only 1 mA of current in the off-hook fully operational state. If necessary, this can be reduced as low as 200  $\mu$ A during loop signalling. In the on-hook condition a bias current of typically only 10nA is required by the CMOS RAM to retain data.

To optimise the design, both hardware and software should be considered together.

The first task is to define the desired sequence of events, construct a flow chart and plan the store and COP420C RAM maps. A protocol similar to the following could be used:

1. A CMOS RAM is mapped into blocks each storing a 16-digit number. There are 16 such blocks in this example (numbered 0-15). Digits are stored in BCD format.
2. A code is defined to be keyed in so that numbers to be written to or read from the store can be differentiated from numbers for direct dialing. For example, using a 3  $\times$  4 keyboard:

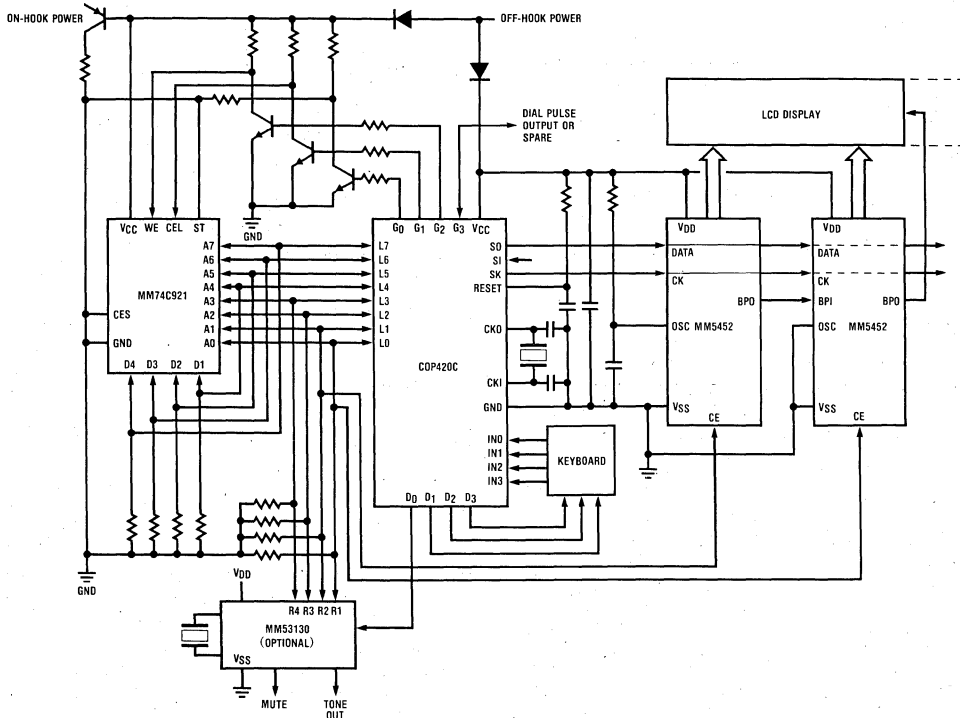


Figure 1. COP420C Telephone-Line Powered Repertory Dialer

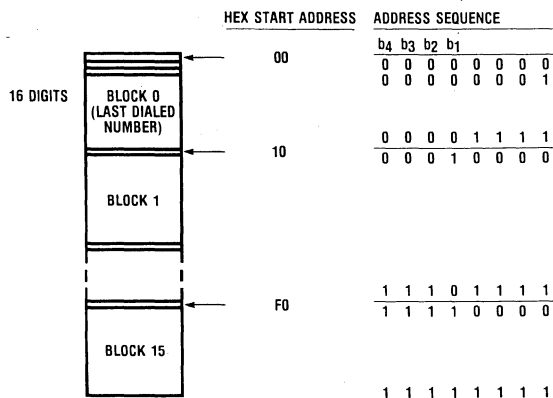


Figure 2. 256 x 4 CMOS RAM Map

- to direct dial a number and enter it into store block 0 (last number dialed): nnn...n
- to automatically dial a number stored in block number 'b': \*b
- to dial a number and write it into store block 'b': nn...n#b
- to enter a number into store block 'b' without dialing out: #nn...n#b

Note that, if there are less than 16 digits in a telephone number, it is necessary to identify the last digit, so that dialing stops at the correct stage. This is conveniently done by using the #b to add a unique code (e.g., 1111) in the memory location after the last digit. Then a test is made for this "stop" code prior to dialing out each digit.

- One 16 x 4 bit register in COP420C RAM(B) is allocated to temporary storage of the telephone number to be processed (say register 0), and two pointers are also set up in one of the other registers. One pointer stores the "nibble" location in register 0 where the next digit (entered from keyboard or store) is to be written into RAM(B); the other stores the "nibble" location in register 0 of the digit currently being dialed out. After dialing out each digit, these write and read pointers are compared to determine if there are further digits to be dialed.
- For block lengths of 16 digits, the storage block number 'b' (expressed in four bit binary e.g., b<sub>4</sub>b<sub>3</sub>b<sub>2</sub>b<sub>1</sub>) conveniently becomes the four most significant bits of the store address of each digit of a telephone number, as shown in Figure 2.

For storage of telephone numbers in block lengths other than 16 digits this method does not allow the maximum packing density of the memory to be achieved. For better efficiency a look-up table should be created in COP420C ROM. This lists the memory addresses of the first digit of each telephone number. The block number 'b' can then be used as the four least significant bits of the ROM table location containing the address of the first digit of block 'b' in the store. This block start address is then fetched from ROM using the LQID instruction, followed by CQMA. The addresses of subsequent digits in a number are easily computed by incrementing the start address.

## CMOS Ram Interface

A 256 x 4 bit CMOS RAM is well suited to the COP420C architecture, and provides conveniently organized storage for 16 telephone numbers of 16 digits each, or similar multiples. The 8 bit bi-directional, TRI-STATE<sup>®</sup> L-port on the COP420C directly drives the RAM address inputs. Several options are available for interfacing to the RAM data input and output ports. A particularly economic solution is available using the 18 pin MM74C921 CMOS RAM, which has common data I/O and three control lines rather than the usual two. This enables the RAM DATA-OUT drivers to be tri-stated during both read and write operations. Thus the DATA-IN/OUT port can be multiplexed on the COP420C L-port without bus contention when the L-port drivers are driving the bus. The RAM data port is connected to L4-L7, which is more convenient than L0-L3 when using the INL instruction to read data from the store. The data is read directly into the RAM(B) location pointed to by Bd.

Outputs G0, G1 and G2 are used here to drive the MM74C921 store control lines, although any of the G or D outputs could be used.

To address the store, DATA-OUT is tri-stated by taking C<sub>EL</sub> high, so that the L-port output drivers control the bus. The address is strobed into the RAM under  $\overline{ST}$  control, then the address is removed enabling the bus to be used for data transfer. Option 00 should be chosen for all L outputs so that the L-port can be tri-stated during a store read operation — the RAM data outputs will drive the bus.

An example routine to write a digit into the store, with correct timing, is shown in List 1. To read a number from store into the COP420C internal RAM(B) takes, typically 30 instruction cycles per digit, or 480 cycles for a 16-digit number. This takes approximately 32ms with a 480kHz clock, divided by 32 option (using a 455kHz low-cost ceramic resonator in anti-resonant mode). Thus, for abbreviated dialing, a complete number can be read from store without significant delay to the start of loop signalling.

For on-hook data retention it is necessary to power the store RAM from a back-up battery or via a 10M $\Omega$  resistor from the telephone line side of the hookswitch. The transistor inverters on the MM74C921 control inputs ensure that stored data is not corrupted as the COP420C powers up or down under hookswitch control.



Pull-down resistors on the address inputs are also recommended to ensure that these inputs do not "float" between the guaranteed input logic levels and cause increased current consumption. 100 kohms minimum must be used for circuit operation down to 3V. Contact your National Semiconductor representative for further assistance on operating the MM74C921 at reduced supply voltages (3 to 5V).

### Memory Expansion

Larger memories, for example a  $1K \times 4$  CMOS RAM, can be used by adding latches which are loaded with the additional address line information via the L, D or G ports. Alternatively, the COP451 RAM Interface Chip generates address capability for 1-bit wide RAMS up to 8K bits. Store data is organized in 64 bit blocks accessed via the COP420C SIO port, leaving the L port free for peripheral chip selects or other functions.

### COP499 CMOS RAM Solution

An alternate scheme, particularly attractive for smaller stores, is to use the COP499 256 bit CMOS RAM with power-down control switch. The RAM is organized as four registers, each of 16 4-bit "nibbles," enabling one 16 digit (or two 8 digit) telephone numbers to be stored per register.

Data transfer between COP420C RAM registers and the store is via the SIO port, at a rate of one bit per instruction cycle. Thus a typical search and read of a 16 digit telephone number takes only a few milliseconds in this system.

### Keyboard Interface

The keyboard interface depends on whether separate keys are to be provided for each stored telephone number or an abbreviated code is used to address a number within a block in the store. The latter method is illustrated here, using a  $3 \times 4$  single-contact keyboard. COP420C outputs D1-D3 are used to scan the columns, and the rows are read by inputs IN0-IN3.

Keyboards as large as  $8 \times 4$  can be directly interfaced by multiplexing the scan on the 8-bit L-port. In this case, each scan line must be diode isolated from other peripheral devices on the L-port to ensure that multiple key closures cannot corrupt data intended for the peripheral. Separating the keyboard from the store interface, as shown here, simplifies keyboard monitoring during loop signalling routines.

Methods of keyboard scanning, debounce and decoding are well documented with software listings, in the COPS™ Family User's Guide, Section 5.3. As keyboard routines are usually the most frequently used of all repertory dialer functions, every effort should be made to maximize their efficiency. For minimum COP420C clock frequency, hence minimum supply current, keyboard scan intervals and debounce times must be as long as possible. Keypads with adequate hysteresis can accept scan intervals as long as 15 or even 20 milliseconds, with only two consecutive scans of a key required for validation.

### Loop-Disconnect Signalling

The design thus far has left pins D0 and G3 still available for use as outputs dedicated to driving interface circuits to make and break the loop current at 10 (or 20) pulses-per-second, and to mute the receiver during outpulsing. These outputs are controlled by software timing routines which generate 60 and 40 millisecond delay loops for a 1.5:1 break/make ratio, or 67 and 33 millisecond delays for a 2:1 break/make ratio. An inter-digit pause interval of, typically, 800 milliseconds is also required.

The easy method of writing these delay routines is to set up loop count constants which are incremented after a fixed number of instruction cycles. These instruction cycles are, of course, put to good use by including keyboard, display or other routines (taking care that no conditional jumps are included which might vary the loop length or, worse still, cause an exit from the loop). NOP (No Operation) instructions may be necessary to achieve the desired loop length, although they do waste valuable ROM space. The oscillator frequency, loop constant and loop length should be calculated together to obtain the desired delays with maximum programming efficiency.

Several other timing methods are described in the COPS™ Family User's Guide, Section 4.5.

These timing schemes are generally implemented with the COP420C running at normal speed and current consumption. As such they are particularly suitable for series mode pulse dialers, where the pulsing loop includes the resistance of the speech network. Some specifications, however, impose additional constraints, such as requiring the pulsing element to shunt the speech network. This results in virtually no voltage available at the telephone terminals to supply the dialer during loop make periods. In addition, the loop current during a loop break period may be specified to not exceed a certain value, sometimes as low as 200 microamps.

Solutions to both of these problems involve careful power supply design, using a capacitor to maintain power to the circuit during dialing interruptions. The current consumption must therefore be reduced to a minimum by using the slowest possible instruction cycle time consistent with the execution of the real-time routines, such as keyboard scanning and signalling.

A method of further reducing COP420C current consumption during dialing is to use the idle mode and on-chip timer to generate loop make and break timing routines. For example, for a 2:1 break/make ratio at 10 pps a timing loop is set up consisting of 8192 clock cycles, which is the period of the COP420C timer. With an oscillator frequency of 480 kHz, divided by 32 as before, the timing loop length turns out to be 17.07 milliseconds. This is an adequate interval between keyboard scans, and four such loops can provide a 68.24 millisecond loop break period. The oscillator frequency can be changed to provide other loop break periods.

Figure 3 illustrates this method.

A digit train is started by setting the MUTE output and timing a pre-pulsing pause if required. Before starting the first break period, an IT, (Idle till Timer) instruction is executed in order to synchronize the break period with

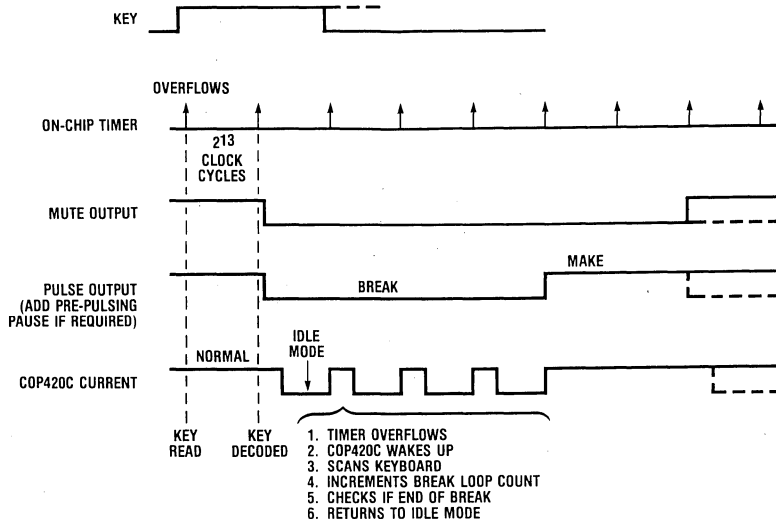


Figure 3. Loop Break Period Timing Using Idle Mode

the start of the 8192 cycle timer. When the timer overflows, the COP420C wakes up and starts the break period timing routine, which will count four 17.07 millisecond timed loops. Then a loop count of Hex C (12) is set in RAM (B) and the D0 or (G3) output set to the loop break state. After another scan of the keyboard the IT instruction puts the COP420C into the low power mode for the remainder of the 8192 clock cycles.

When the timer next overflows, the COP420C wakes up, scans the keyboard and then interrogates the loop count to see if the end of the break period has arrived. If not, the loop count is incremented by one and another IT instruction puts the COP420C into the low power mode for the remainder of the Timer period.

After four of these loops, i.e. 68.24 milliseconds, the loop count is Hex F, and therefore overflows and sets the carry when incremented. A Skip on Carry (SKC) instruction then exits the break period timing routine and enters a loop make period routine of two 17 millisecond timed periods.

Timing routines such as this, using the idle mode, reduce the COP420C mean  $I_{DD}$  current by typically 50%. As no code is executed during idle mode timing, operations such as memory access and display updating are inserted during inter-digital pauses. For larger systems, which may require too many real time routines to enable idle mode timing to be used, a dual clock option is available. The D0 output becomes a fast RC controlled oscillator for executing most routines, then for accurate timing at reduced power consumption the CKO oscillator is used with a low frequency crystal, e.g., a 32 kHz watch crystal.

The several timing methods available on the COP420C enable a wide variety of system configurations to be designed which can satisfy all the constraints of a telephone line-powered pulse dialer.

## Dual-Tone Multi-Frequency Signalling

The standard form of fast 2-of-8 tone signalling is easily implemented by adding a device such as the National Semiconductor MM53130 Touch Tone™ generator to the L port. This device, which normally directly scans the keyboard itself, has a pin selectable option to accept binary coded data on the four row inputs. It is shown in Figure 1 connected to outputs L<sub>0</sub>-L<sub>3</sub>. Timing loops, generated in the same way as those for pulse dialing, control the duration of output tone generation by connecting the Tone Disable pin to the COP420C D0 output.

The MUTE function is generated by the MM53130 itself, so that G3 becomes available for use as a general purpose bidirectional I/O pin. Possible uses include reading a strap to logic "1" or "0" to determine whether particular facility routines are required; raising a status indication (following on-hook dialing, for example); or chip-enable for some additional peripheral device.

## Display

Recent improvements in liquid crystal display technology, together with new integrated driver circuits, make LCD a suitable choice for a reliable low-power display. This enables not only the dialed number to be displayed, but also the addition of facilities such as clock, timer and calculator if desired.

Low cost drivers for both multiplexed and direct displays are easily added to COP™ systems using the MICRO-WIRE™ interface. This uses the SIO register and SK output to serially clock data into various peripheral devices. The COP472 display driver, for example, decodes serial display and control data and provides multiplexed drive for 12 segments on each of 3 back-planes e.g. 4½ digits.

While multiplexed LCD's work well in 5 to 12V systems, reduced contrast at lower voltages tends to make direct drive preferable. For operation down to 3V in this system the MM5452 32 segment driver is used. Like all MICRO-WIRE™ compatible peripherals, multiple devices can be cascaded by providing each with a separate CHIP ENABLE line for unique addressing. The desired CHIP ENABLE is taken low prior to clocking in 32 bits of data (one per segment) plus four additional clock pulses for internal control. No further action is required until it is desired to change display data, as display refreshing is independently controlled by the on-chip oscillator. Furthermore, because clock and data are internally gated together in the MM5452 the CHIP ENABLE inputs can be multiplexed on the COP420C L-port provided the SK clock is kept at static logic '0' when not updating the display. Thus, up to 8 display drivers or other peripherals can be driven via the MICROWIRE™ interface without the need for additional decoders.

## Conclusion

Many other variations are possible on the repertory dialer scheme presented here. Call timing routines can be added; so can a real time clock if an on-hook power source is available; strapping fields for option selection can be read via the serial input SI, or can be multiplexed on the L-port. The L-port, in particular, is seen to be extremely flexible both in hardware and software capability, enabling a minimum device count system to be built without the need for I/O expanders. Use of the on-chip timer enables all the specifications for a telephone line powered repertory dialer to be met. If local power is available, the NMOS versions of the COPS™ family can also be considered.

Where mounting space is a problem, National Semiconductor can offer a low-cost custom module solution, whereby the devices are attached in die form to a circuit board, along with the display. Consult your local National Semiconductor Sales office for details.

COP CROSS ASSEMBLER PAGE: 1  
REPEX

```

1          . TITLE REPEX
2          ; EXAMPLE ROUTINE TO WRITE THE DIGIT IN REGISTER 0, 0
3          ; INTO THE FIRST ADDRESS SPACE IN STORE BLOCK 1.
4          ; TELEPHONE NUMBERS OF 20 DIGITS ARE CHOSEN TO
5          ; DEMONSTRATE THE USE OF A LOOK-UP TABLE TO GENERATE
6          ; THE START ADDRESS. THUS, BLOCK 1 START ADDRESS IS
7          ; HEX 15, WHICH WILL BE STORED IN A LOOK-UP TABLE
8          ; BEGINNING IN ROM LOCATION OF 1 (MUST BE IN SAME 4
9          ; PAGE SEGMENT OF ROM)
10
11
12 000 00          WRITE:  CLRA
13 001 3350        OGI      0
14 003 2E          LBI      2, 15          ; POINT TO BLOCK NO. IN RAM (B)
15 004 5F          AISC     15          ; LOAD F INTO ACCUMULATOR
16 005 BF          LQID     ; CONTENTS OF F1 PUT IN Q
17 006 2D          LBI      2, 14          ; WANT ADDRESS COPIED IN RAM (B)
18 007 332C        CQMA     ; ADDRESS UPPER NIBBLE IN 2, 14
19 009 2C          LBI      2, 13
20 00A 06          X        ; ADDRESS LOWER NIBBLE IN 2, 13
21 00B 3364        LEI      4          ; ADDRESS OUT TO 74C921
22 00D 3357        OGI      7          ; ST, WE, CEL GO LO
23 00F 2320        LDD      2, 0        ; PUTS FIRST DIGIT IN ACC
24 011 333C        CAMQ     ; DIGIT OUT TO 74C921 DATA PORT
25 013 3350        OGI      0          ; DATA STROBED IN
26 015 3360        LEI      0          ; TRI STATE L
27          . = X/F1
28 0F1 15          .WORD X'15
29          END

```

# The COP444L Evaluation Device 444L-EVAL

National Semiconductor  
COP Note 4  
Leonard A. Distaso  
February 1981



The 444L-EVAL is a preprogrammed COP444L intended to demonstrate operating characteristics and facilitate user familiarization and evaluation of the COP444L and the COPSTM family in general.

The 444L-EVAL has two mutually exclusive operating modes: an up/down counter/timer or a simple music synthesizer. The state of pin L7 at power up determines the operating mode.

## 1. The 444L-EVAL as a Simple Music Synthesizer

Figure 1 indicates the connection of the 444L-EVAL as a simple music synthesizer. As the diagram indicates, the

connections required for operation are minimal. The oscillator may be a crystal circuit using CKI and CKO; an external oscillator to CKI; or an RC network using CKI and CKO. As should be expected, the crystal circuit provides the greatest frequency stability and precision. The RC network will provide an acceptable oscillation frequency but that frequency will be neither precise nor stable over temperature and voltage. The external oscillator, of course, is as good as its source. The frequencies for the various notes and delay times are set up assuming that the oscillator frequency is 2MHz. Three modes of operation are available in the music synthesizer mode: play a note; play one of four stored tunes; or record a tune for subsequent replay.

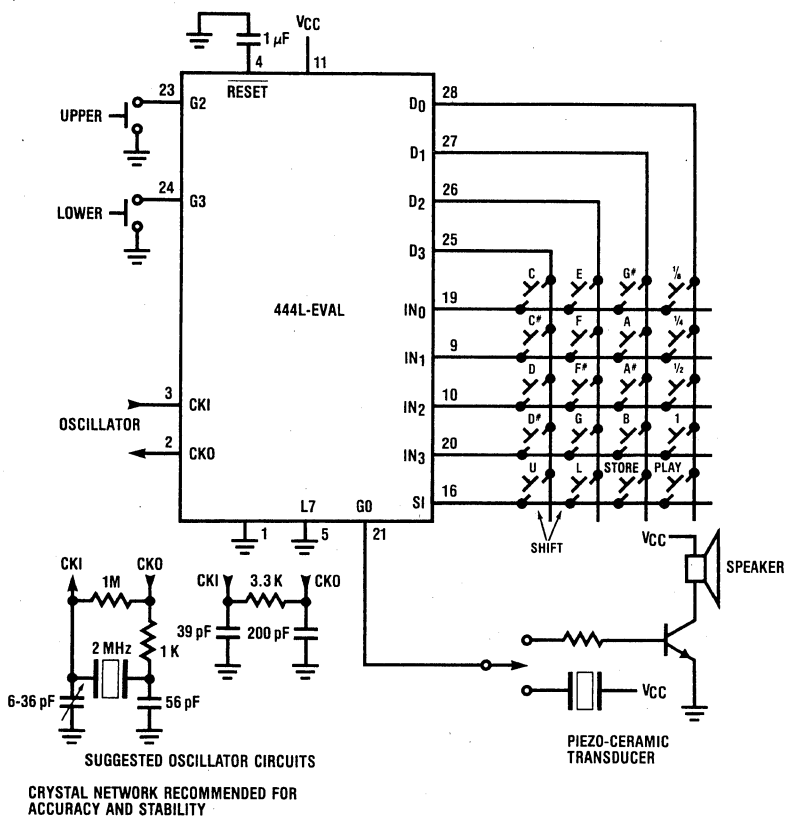


Figure 1. 444L-EVAL as simple Music Synthesizer

## 1.A. Play a Note

Twelve keys, representing the twelve notes in one octave, are labeled "C" through "B". Depressing a key causes a square wave of the corresponding frequency to output at GO. The user may drive a piezo-ceramic transducer directly with this signal. With the appropriate buffering, the user may use this signal to drive anything he wishes. A simple transistor driver is sufficient to drive a small speaker. The user can be as simple or as complex as he desires at this point — e.g. he can do some wave shaping, add an audio amplifier, and drive a high quality speaker.

The 444L-EVAL has a range of two and one-half octaves: the basic octave on the keyboard (which is middle C and the 11 notes above it in the chromatic scale), one full octave above the basic octave and one-half octave below the basic octave. The notes in the basic octave are played by depressing the appropriate key (one key at a time — the keyboard has no rollover provisions). A note in the upper octave is played by first depressing and releasing the U SHIFT key and then depressing the note key. Similarly, a note in the lower one-half octave is played by first depressing and releasing the L SHIFT key and then depressing the note key. Two other shift keys are present: UPPER and LOWER. All notes played while the UPPER key is held down will be in the upper octave. Similarly, notes F# through B when played while the LOWER key is held down will be in the lower one-half octave. The lower octave notes C through F are not present and depressing any of these 6 keys while the LOWER key is held down or after depressing the L SHIFT key will play the note in the basic octave.

## 1.B. Play Stored Tune

The 444L-EVAL can play four preprogrammed tunes. Depressing PLAY followed by "1/8", "1/4", "1/2", or "1" will cause one of these tunes to be played. The tunes are:

- PLAY 1 — Music Box Dancer
- PLAY 1/2 — Santa Lucia
- PLAY 1/4 — Godfather Theme
- PLAY 1/8 — Theme from Tchaikowsky  
Piano Concerto #1

## 1.C. Record A Tune

Any combination of notes and rests up to a total of 48 may be stored in RAM for later replay. A note is stored by depressing the appropriate key(s), followed by the duration of the note (1/8 note, 1/4 note, 3/8 note, 1/2 note, 3/4 note, whole(1) note), followed by STORE. A rest is stored by selecting the duration and depressing STORE. The rests or durations of 1/8, 3/8, 1/2, and 3/4 are obtained by first depressing L SHIFT and then 1/8, 1/4, 1/2, or 1 respectively. When the tune is complete press PLAY followed by STORE. The tune will be played for immediate audition. Subsequent depression of PLAY and then STORE will play the last stored tune.

Only one tune may be stored, regardless of length. Attempts to store a new or second tune will erase the previously stored tune. There are no editing features in this mode. (In a "real system" of this type some form of

editing would be desirable. It would not be difficult to add editing features.)

NOTE: The accuracy of the tones produced is a function of the oscillator accuracy and stability. The crystal oscillator, or an accurate, stable external oscillator is recommended.

## 2. The 444L-EVAL as an Up/Down Counter/Timer

By connecting pin L7 to V<sub>CC</sub> and providing power and oscillator the 444L-EVAL functions as an 8 digit binary/BCD up/down counter. In addition, an approximate 1Hz signal is produced by the device. The 444L-EVAL can drive a single digit LED display directly. With the appropriate driver (COP472, COP470, MM5450/5451) the device can drive a 4 digit LCD, VF, or LED display. Any combination of these displays can be connected at any given time.

The binary/BCD and up/down modes are controlled by the states of input pins IN0 and IN2 as indicated below:

IN0 = 1 (Default state)	— BCD counter
IN0 = 0	— Binary Counter
IN2 = 1 (Default state)	— Count Up
IN2 = 0	— Count Down

The up/down control may be changed at any time. Changing the binary-BCD control during operation clears the counter before counting begins in the new mode.

Pins G2 and G3 provide display control to the user. He can choose to view either the most significant 4 digits of the counter or the least significant 4 digits of the counter. Further, the user can disable the update of the 4 digit displays. The controls are as follows:

G2 = 1 (Default state)	— Enable update of 4 digit displays
G2 = 0	— Disable update of 4 digit displays
G3 = 1 (Default state)	— Display least significant 4 digits of counter
G3 = 0	— Display most significant 4 digits of counter

The single digit LED display displays the least significant digit of the counter. (Note, the direct drive capability for the single digit LED display refers to a small LED digit — NSA1541A, NSA1166, or equivalent.)

## 2.A. I/O Mode

The 444L-EVAL has the capability to allow the user to read or write the 8 digit counter through the L port. In the I/O mode, the single digit LED display is disabled. The 4 digit displays are not affected. In this mode pins D0 and IN3 are used for the handshaking sequence. D0 is a Ready/Write signal from the 444L-EVAL to the outside; IN3 is a Write/Acknowledge from the outside to the 444L-EVAL. Data I/O is via L0-L3 with L0 being the least significant bit. Data is standard BCD for the BCD counter mode or standard hex for the binary counter mode. The digit address is on pins L4-L6 with L4 being

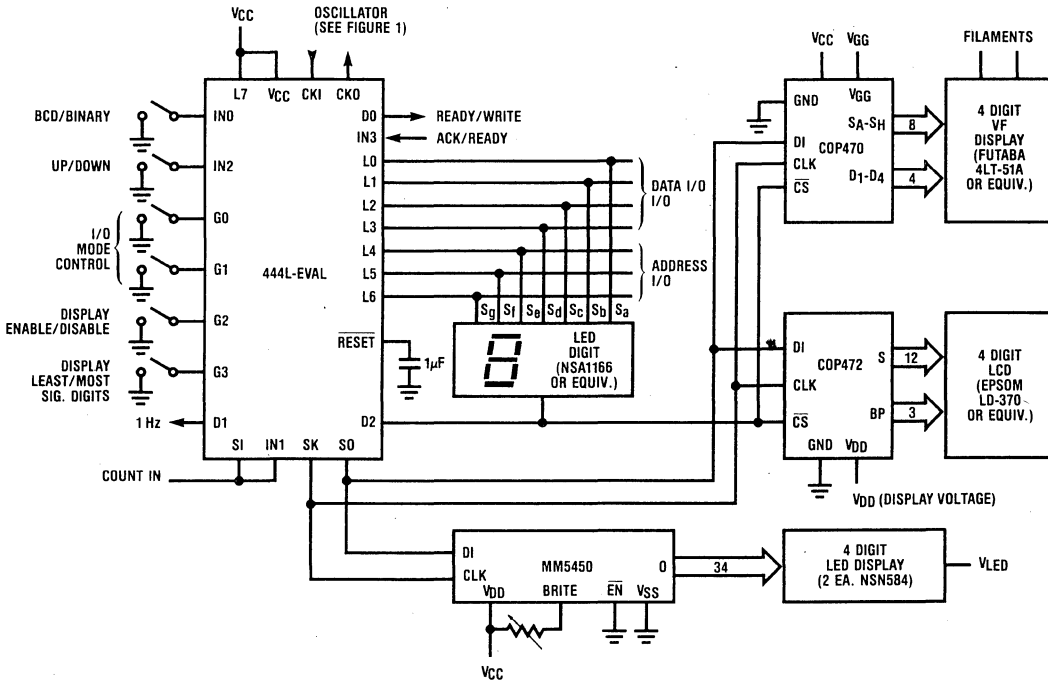


Figure 2. 444L-EVAL in Counter Mode

the least significant bit. Digit address 0 is the least significant digit of the counter; digit address 7 is the most significant digit of the counter. The I/O modes are controlled by pins G0 and G1 as follows:

G0	G1	Description
0	0	Output data with handshake, single digit LED off
0	1	Input data with handshake, single digit LED off
1	0	Auto output, no handshake, single digit LED on
1	1	Default condition, No I/O, single digit LED displays least significant digit of counter

### 2.A.1. Output Data with Handshake

With this mode selected the 444L-EVAL will output data with a handshake sequence. Note that the outputting of data is relatively slow as the device is counting and updating displays between successive digit outputs.

Before data is output, or the next digit of the counter is output, the 444L-EVAL must see IN3 (Acknowledge or ready from the external world high). The Ready/Write pin (D0) is assumed to be high at this point. With D0 high and IN3 high, the device will output the data and digit address. After the data and address are output, the D0 line — functioning as a write strobe here — goes low. The 444L-EVAL then expects the signal at IN3 to go low indicating that the external world has read the data. When the device sees IN3 go low, D0 will be brought high indicating that the sequence is ready to repeat as soon as IN3 goes high again. The counter digits are out-

put sequentially from least significant digit (digit address 0) through most significant digit (digit address 7). The sequence will continuously repeat as long as this mode is selected.

### 2.A.2. Input Data with Handshake

The 444L-EVAL will take data supplied to it and load the counter. The sequence is similar to that described above for the output mode. The external device(s) supplies both the data and the digit address where that data is to be loaded.

When sending data to the 444L-EVAL, the external circuitry must test that the device is ready to receive data (D0 high). Then the data and address should be presented at the L port. Then the Write signal (IN3) should be driven low. The 444L-EVAL will read the data and then drive D0 low. When D0 goes low, the external circuitry should bring IN3 high. After IN3 returns high, the 444L-EVAL will signal it is ready to receive data by sending D0 high. Note that this sequence is relatively slow. The 444L-EVAL is performing several operations between successive read operations.

### 2.A.3. Automatic Output Mode

In the automatic output mode, the single digit LED is on. It is not displaying the least significant digit of the counter in this mode. The display is on so that the user can connect this LED digit, select the automatic output mode, and observe the states of the L lines without having to put more sophisticated equipment or circuitry external to the 444L-EVAL. Segments a through d are

pins L0 through L3; segments e, f, g are pins L4, L5, and L6. Thus the user can observe the digit address changing and observe the corresponding data.

In this mode, the state of pin IN3 is irrelevant. The 444L-EVAL sequentially outputs the digits of the coun-

ter. D0 goes high when the data and address is being changed. D0 goes low when the data is valid. As in the other I/O modes, the process is slow. There is about 4 to 5 milliseconds between the successive digit outputs.

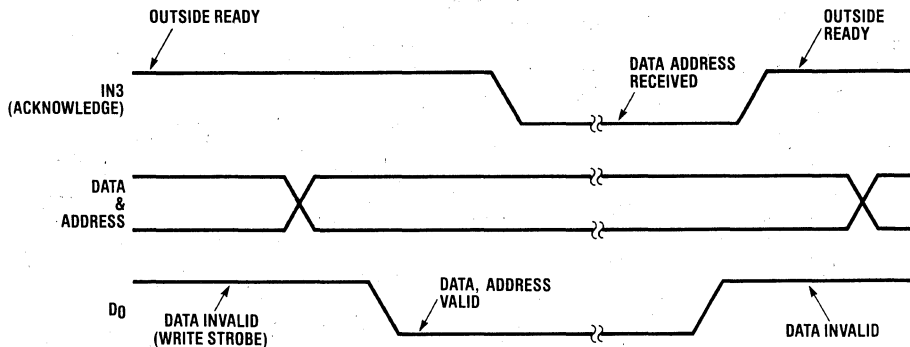


Figure 3A. Relative Timing — Output Handshake.

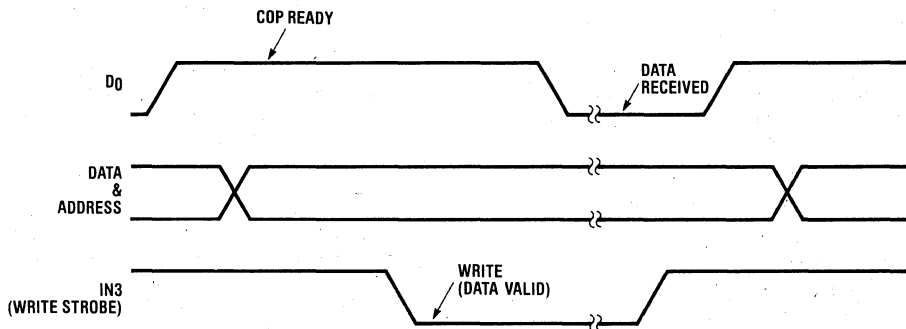


Figure 3B. Relative Timing — Input Handshake

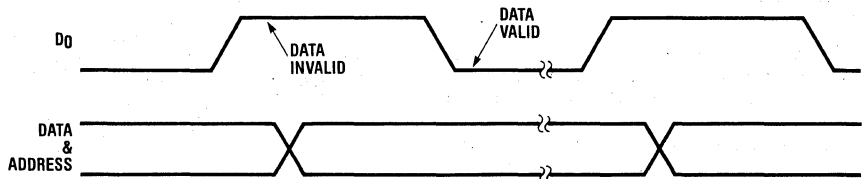


Figure 3C. Relative Timing — Automatic Output

### 3. Selected Options

The 444L-EVAL has the following options selected:

GND	Option 1 = 0	
CKO	Option 2 = 0	CKO is clock generator output to crystal
CKI	Option 3 = 0	CKI oscillator input divide by 32
RESET	Option 4 = 0	Load device to $V_{CC}$ on RESET
L7	Option 5 = 0	Standard output on L7
L6	Option 6 = 2	High current LED direct segment drive on L6
L5	Option 7 = 2	High current LED direct segment drive on L5
L4	Option 8 = 2	High current LED direct segment drive on L4
IN1	Option 9 = 0	Load device to $V_{CC}$ on IN1
IN2	Option 10 = 0	Load device to $V_{CC}$ on IN2
$V_{CC}$	Option 11 = 1	4.5V to 9.5V operation
L3	Option 12 = 2	High current LED direct segment drive on L3
L2	Option 13 = 2	High current LED direct segment drive on L2
L1	Option 14 = 2	High current LED direct segment drive on L1
L0	Option 15 = 2	High current LED direct segment drive on L0
SI	Option 16 = 0	Load device to $V_{CC}$ on SI
SO	Option 17 = 2	Push-pull output on SO
SK	Option 18 = 2	Push-pull output on SK
IN0	Option 19 = 0	Load device to $V_{CC}$ on IN0
IN3	Option 20 = 0	Load device to $V_{CC}$ on IN3
G0	Option 21 = 0	Very high current standard output on G0
G1	Option 22 = 2	High current standard output on G1
G2	Option 23 = 4	Standard LSTTL output on G2
G3	Option 24 = 4	Standard LSTTL output on G3
D3	Option 25 = 0	Very high current standard output on D3
D2	Option 26 = 0	Very high current standard output on D2
D1	Option 27 = 0	Very high current standard output on D1

D0	Option 28 = 0	Very high current standard output on D0
	Option 29 = 0	Standard TTL input levels on L
	Option 30 = 0	Standard TTL input levels on IN
	Option 31 = 0	Standard TTL input levels on G
	Option 32 = 0	Standard TTL input levels on SI
	Option 33 = 1	Schmitt trigger inputs on RESET
	Option 34 = 0	CKO input levels, not used here
	Option 35 = 0	COP444L
	Option 36 = 0	Normal RESET operation

### 4. Conclusion

The 444L-EVAL demonstrates much of the capability of the COP444L. It does not indicate the limits of the device by any means. The I/O features were included to demonstrate that capability. The fact that they are slow is due strictly to the program. If such I/O capability were a necessary part of an application it could be accomplished much much faster than was done here. The counter modes are quite versatile and are generally self explanatory. It was fairly easy to provide a counter with the versatility of that included here. The music synthesis mode demonstrates clearly the program efficiency of the device.

The 444L-EVAL is intended for demonstration. There is no question that aspects of its operation could be improved and tailored to a specific application. It is unlikely that this particular combination of features would be found in any one application. It is also interesting to note that the program memory in the device is not full. There is still a significant amount of room left in the ROM. This should serve to make it clear that the capabilities of the device have not been stretched at all in order to include these demonstration functions.



# Oscillator Characteristics of COPS™ Microcontrollers

National Semiconductor  
COP Note 5  
February 1981



## Table of Contents

I. Introduction	9-128
II. RC Oscillator Option	9-128
III. Crystal or Inverter Option	9-129
A. COP420/COP402	9-129
1. RC Networks	9-129
2. L, LC, and RLC Networks	9-129
B. COP420L	9-130
C. COP420C	9-130
D. COP410L	9-130
E. General Notes	9-130
IV. Conclusion	9-130

COPS is a trademark of National Semiconductor Corp.

## I. Introduction

COPS™ microcontrollers will operate with a wide variety of oscillator circuits. This paper focuses on two of the oscillator options available on COPS microcontrollers: the internal RC oscillator, and the crystal or inverter oscillator. The typical behavior of the RC oscillator with temperature and voltage (and typical values of R and C) is documented. For the crystal or inverter option, circuit configurations (RC, RL, RLC, R, LC, L) are presented which will allow the microcontroller to operate properly without the use of ceramic resonator or crystal.

The data contained here was obtained from a number of devices picked at random from production runs. The passive components used were inexpensive, uncompensated devices: standard carbon resistors, ceramic or foil capacitors, and air core or iron core inductors. To provide reasonably clear data on the characteristics of the microcontroller itself, no attempt at compensation for the external components was made.

## II. RC Oscillator Option

With the RC oscillator option selected, the graphs in Figures 1 through 6 indicate the variation of the instruction cycle time of the microcontroller with temperature and voltage. Typical R and C values, as recommended in the respective device data sheets, were used. The graphs are composite graphs reflecting the worst case variations of the devices tested. Therefore, the graphs show a percentage change of the instruction cycle time from a base or reference value. Where the results are plotted against voltage the reference is the value at  $V_{CC}=5$  volts. Where the results are plotted against temperature, the reference is the value at  $T=20^{\circ}\text{C}$ . A positive percent variation indicates a longer instruction cycle time and therefore a slower oscillator frequency. Similarly, a negative percent variation indicates a shorter instruction cycle time and therefore a faster oscillator frequency.

The measurements were taken by holding the RESET pin of the device low and measuring the period of the waveform at pin SK. In this mode the SK period is the instruction cycle time. The oscillator frequency is given by the following:

$$\text{frequency} = \frac{4}{\text{SK period}}$$

Measurements were taken at temperatures between  $-40^{\circ}\text{C}$  and  $+85^{\circ}\text{C}$  and at  $V_{CC}$  values between 4.5 volts and 9.5 volts. However, the reader must remember that the COP400 series is specified only between  $0^{\circ}\text{C}$  and  $+70^{\circ}\text{C}$ . The reader must also remember that the COP420 is specified at  $V_{CC}$  levels between 4.5 volts and 6.3 volts only. The data here is usable for the COP300 series, which is specified at the extended temperature range of  $-40^{\circ}\text{C}$  to  $+85^{\circ}\text{C}$ . However, the reader must keep in mind the generally more restricted  $V_{CC}$  range for some of the various COP300 series microcontrollers.

The graphs in Figures 1 through 6 reflect the variation of the microcontroller only. The resistor and capacitor were not in the temperature chamber with the COPS™ device. Obviously, the results will be affected by the variation of the R and C with temperature. However, this can vary dramatically with the type of components used. The user will have to combine the data here with the characteristics of the external components used to determine what type of variation may be expected in his system.

### III. Crystal or Inverter Option

With the crystal or inverter option selected on the COPSTM microcontroller there is, effectively, an inverter between the CKI and CKO pins. CKI is the input to the inverter and CKO is the output. Various passive circuits were connected between CKI and CKO and the results documented. Of the operational circuits, a subset was tested over temperature with the microcontroller only in the temperature chamber. A smaller subset was tested over temperature with both the microcontroller and the oscillator network in the temperature chamber.

The data with the oscillator network in the temperature chamber is obviously highly dependent on the particular components used. This data was taken with standard, inexpensive, uncompensated components. Neither high precision nor high stability components were used. This data is included only to provide the user with some very general indication of how the oscillator frequency may vary with temperature in a real system.

#### III.A. COP420/COP402

Except for the ROM, the COP420 and COP402 are equivalent devices. The internal circuitry of each device is identical. Therefore, data taken for one of the devices is equally applicable to the other. The following discussion will refer to the COP420 but all such references apply equally well to the COP402. Similarly, the graphs for the COP420 apply to the COP402 and *vice versa*.

With the crystal option selected, the COP420 oscillator circuitry will readily oscillate with almost any circuit configuration between CKI and CKO. What difficulty there is lies in finding the network of the device. With the appropriate divide option selected, oscillator frequencies between 800 kHz and 4 MHz are valid for the COP420. No data was taken for any network that produced an oscillation frequency outside the valid range.

##### III.A.1. RC Networks

No single R or single C was found that produced a valid oscillation frequency. The RC network of Figure III.1 was the simplest RC network that produced a valid frequency. With R between 1 k $\Omega$  and 3 k $\Omega$  and C between 0.001  $\mu$ F and 0.005  $\mu$ F oscillation frequencies, at room temperature, of between 3.4 MHz and 4 MHz were observed. Smaller values of C produced higher oscillation frequencies.

With the network of Figure III.1, the oscillation frequency was approximately monotonically decreasing with increasing temperature. Since the oscillation frequencies produced by this network are near the upper end of the valid range, some care should be exercised when using this configuration, especially in an environment where the temperature will go below room temperature.

The addition of capacitor C2, as shown in Figure III.2, both slows down the oscillation frequency and gives greater control over that frequency. With R = 1.5 k $\Omega$  and C1 = 0.005  $\mu$ F, varying C2 from 10 pF to 400 pF produced oscillation frequencies between about 1 MHz and 3.1 MHz. The larger C2 is, the slower the oscillation frequency. The oscillation frequency was monotonically decreasing with increasing temperature.

Figure III.3 adds resistor R2 to the network. Acceptable results were obtained but the network, at least with the values used, did not appear to present any advantage over the network in Figure III.2. With R1 = R2 = 1 k $\Omega$  and C1 = C2 = 100 pF, the frequency was about 3.4 MHz. With R1 = R2 = 2 k $\Omega$ , C1 = 0.001  $\mu$ F, and C2 = 27 pF, the frequency was about 2.9 MHz.

The RC networks provide a reasonably easy and inexpensive means to provide an oscillator for the COP420. As is evident from the graphs, however, the oscillation frequency can vary widely over temperature. The application will determine if that wide variation is acceptable. The configuration of Figure III.2 is the recommended RC network for use in this manner.

#### III.A.2. L, LC, and RLC Networks

Various L, LC, and RLC networks were connected with varying results. Certain networks produced results much more stable than the RC networks; others were no better than the RC networks. With a single inductor connected between CKI and CKO, frequencies between about 1 MHz and 4 MHz were easily obtained. However, the input gate capacitance at CKI (typically 5 pF to 10 pF) and the series resistance of the inductance become factors that impact the oscillation frequency and its stability over temperature.

The addition of a capacitor between CKI and ground tends to reduce the effects of the internal gate capacitance. For the single L, single C network of this type, the capacitor value should be greater than about 50 pF to begin to effectively swamp out the effects of the input gate capacitance. As might be expected, LC combinations which had their resonant frequencies within the valid COP420 frequency range produced the best results.

The addition of another capacitor(s) to the basic two-component LC network, as shown in Figure III.4, produced very good results. Varying the capacitor values in these networks — especially those capacitors between CKI and ground and CKO and ground — provided a great deal of control over the oscillation frequency. In Figure III.4.a, varying C1 from 25 pF to 0.01  $\mu$ F produced oscillation frequencies between about 3 MHz and 1.6 MHz (C2 = 25 pF, L = 56  $\mu$ H). In Figure III.4.b, with C1 = 330 pF, L = 56  $\mu$ H, and C2 = 27 pF, varying C3 between 10 pF and 0.003  $\mu$ F produced oscillation frequencies between about 2 MHz and 1.1 MHz. Varying C2 in Figure III.4.c produced a similar kind of control.

As the graphs indicate, various types of RLC networks were also tried. The range of possible usable circuits here is limited only by the user's imagination and his favorite type of RLC oscillator circuit. When their resonant frequency is within the valid frequency range of the COP420, LC and RLC networks can be a very effective substitute for a crystal. The only potential problem is that a good RLC, or even LC, oscillator circuit may not be a cost-effective substitute for a crystal in a COP420 system. The user will have to make that determination.

COPS is a trademark of National Semiconductor Corp.

### III.B. COP420L

The valid input frequency range for the COP420L, with the appropriate divide option selected, is between 200kHz and 2.097 MHz. With the crystal option selected the COP420L oscillated much less readily than the COP420. No RC circuit of the configuration of Figure III.1 was found that worked acceptably. The circuit of Figure III.2 should be viewed as the minimum RC network that can be used to provide the oscillator to the COP420L when the crystal option is selected. With  $C1 = 39\text{pF}$  and  $C2 = 200\text{pF}$ , varying R from  $2.4\text{k}\Omega$  to  $4.3\text{k}\Omega$  gave oscillation frequencies between 2MHz and 1.3MHz.

The LC networks gave outstanding results with the COP420L. With the simple two-component LC network shown in the graphs, holding C at  $50\text{pF}$  and varying L from  $200\mu\text{H}$  to  $700\mu\text{H}$  gave oscillation frequencies from about 2MHz to 1MHz. Holding L at  $390\mu\text{H}$  and varying C from  $10\text{pF}$  to  $700\text{pF}$  gave oscillation frequencies of about 2MHz to 1.6MHz. Similar results were obtained when a capacitor was placed in parallel with the inductance.

### III.C. COP420C

With the appropriate divide option selected and under the appropriate  $V_{CC}$  values, the COP420C has a valid input frequency range of 32kHz to 2.097 MHz. With the crystal option selected, the COP420C does not oscillate readily when a crystal is not used. No simple RC network was sufficient to make the device oscillate. However, outstanding results were achieved with the LC networks. The graphs are self-explanatory. The networks indicated there produced oscillation frequencies between about 1.8MHz and 800kHz.

### III.D. COP410L

The COP410L has a valid input frequency range of 200kHz to 530kHz. No circuit of the configuration of Figure III.1 produced acceptable results. Figure III.2 is the minimum RC network that should be used with the COP410L in place of the resonator. With  $C1 = 0.001\mu\text{F}$  and  $C2 = 0.002\mu\text{F}$ , varying R from  $3\text{k}\Omega$  to  $12\text{k}\Omega$  gave oscillation frequencies of about 460kHz to 290kHz.

The LC networks also gave very good results. With the simple LC network shown in the graphs, holding L at  $4700\mu\text{H}$  and varying C from  $25\text{pF}$  to  $0.003\mu\text{F}$  gave oscillation frequencies of about 460kHz to 225kHz.

### III.E. GENERAL NOTES

With the crystal or inverter option selected on COPST<sup>TM</sup> microcontrollers, a wide variety of networks may be used in place of the ceramic resonator or crystal. The simple RC network of Figure III.1 will work for the COP420 and COP402. Figure III.2 is the minimum RC network that will work for the COP420L and COP410L (and is also the recommended network for the COP420 and COP402). No RC network is usable in the COP420C in place of the crystal. The RC networks can be expected to have significant variation over temperature and, to a generally somewhat lesser extent, over voltage. If this variation is not acceptable, alternate networks are required.

LC and RLC networks can be used in any of the devices. Appropriately designed, these networks will provide a stable oscillation frequency for the microcontroller. The user will have to allow for the variation of the external components with temperature when using these networks. The problem with networks such as these is that they may not be cost-effective alternatives to the crystal or resonator, especially if high stability, temperature compensated components are used. The user will have to make the determination of cost-effectiveness.

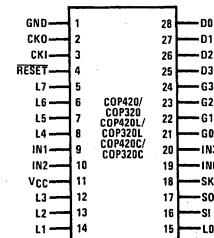
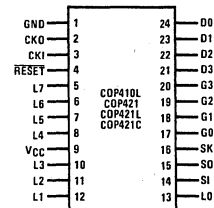
A final note is that all of these networks place a load on the CKO output. If the signal from CKO is needed elsewhere in the system and a circuit similar to one of those discussed in this document is used, it will probably be necessary to buffer the CKO output.

## IV. Conclusion

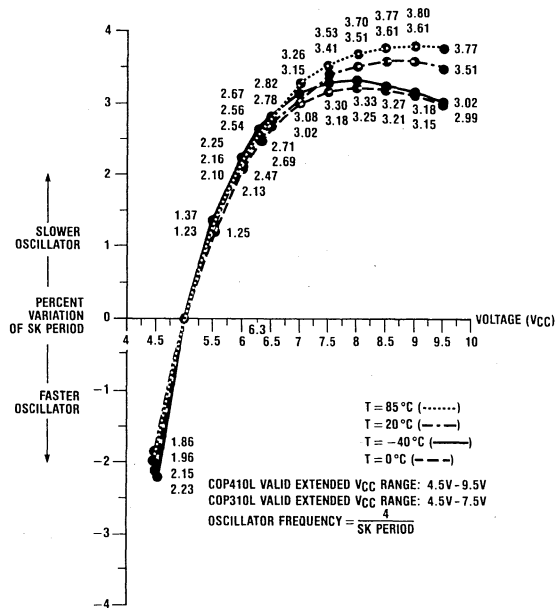
The data contained here does not necessarily indicate the worst case characteristics of any of the microcontrollers involved; and, although the information may be reasonably viewed as representative, National Semiconductor does *not* guarantee that all COPST<sup>TM</sup> microcontrollers will exhibit the characteristics described in this document. This data should not be used as the basis of a system design. In the case of the crystal or inverter option data, the networks described are not necessarily the only ones usable or even the best ones usable. The networks described are generally simple and inexpensive and have all been observed to be functional.

The data contained here is not device characterization data, but is intended solely as a guide for the designer. It provides him with greater flexibility in the oscillator selection in a COPST<sup>TM</sup> system and gives some general indication as to what may be expected with the various circuits described.

COPS is a trademark of National Semiconductor Corp.



COP Microcontroller Pinouts

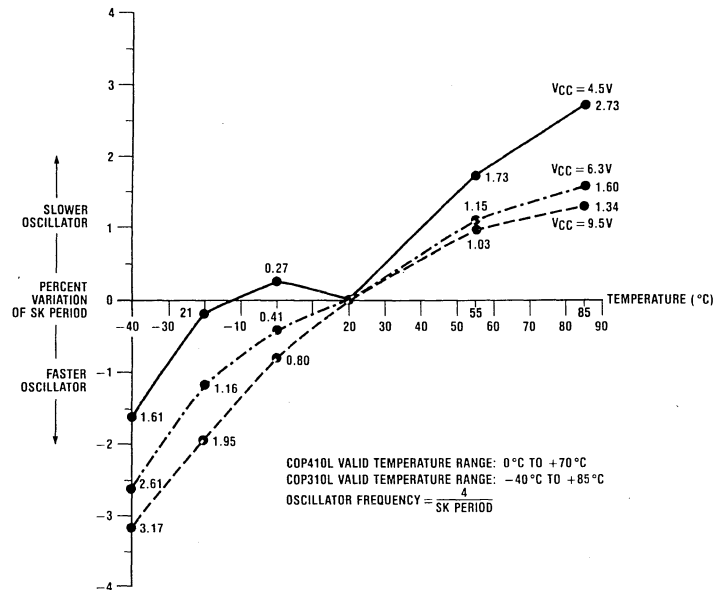


NOTE 1: BASE PERIOD AT VCC = 5.0V.

NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE RC VARIATION WITH TEMPERATURE.

NOTE 3: SK PERIOD = INSTRUCTION CYCLE TIME.

Figure 1. COP310L/COP410L RC Oscillator Variation with VCC

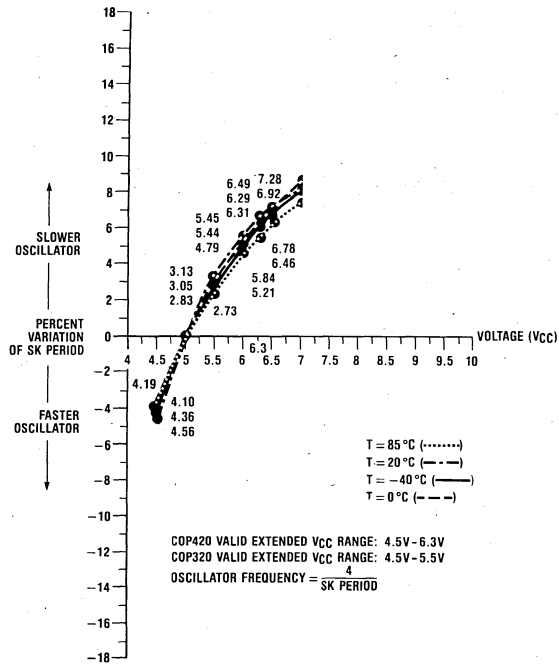


NOTE 1: 20°C = BASE PERIOD.

NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE RC VARIATION WITH TEMPERATURE.

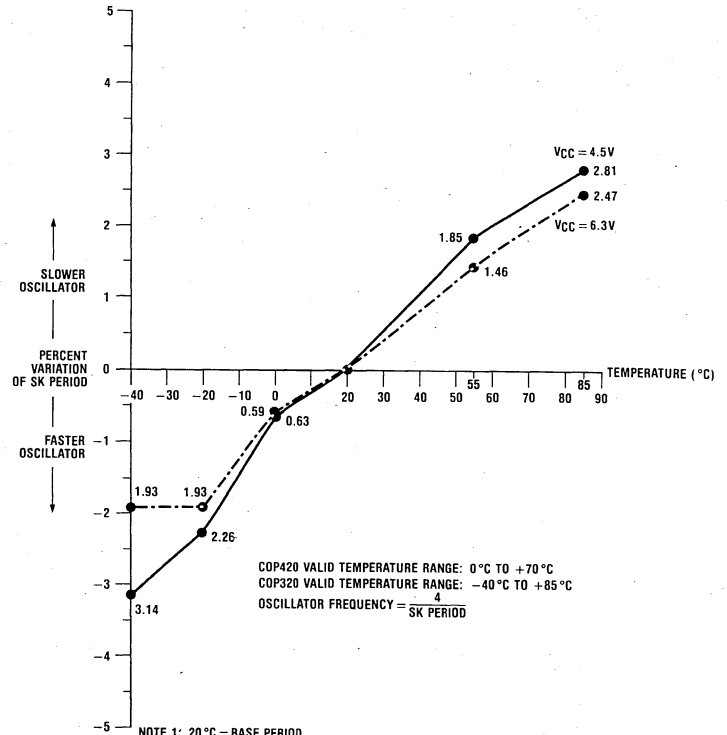
NOTE 3: SK PERIOD = INSTRUCTION CYCLE TIME.

Figure 2. COP310L/COP410L RC Oscillator Variation with Temperature



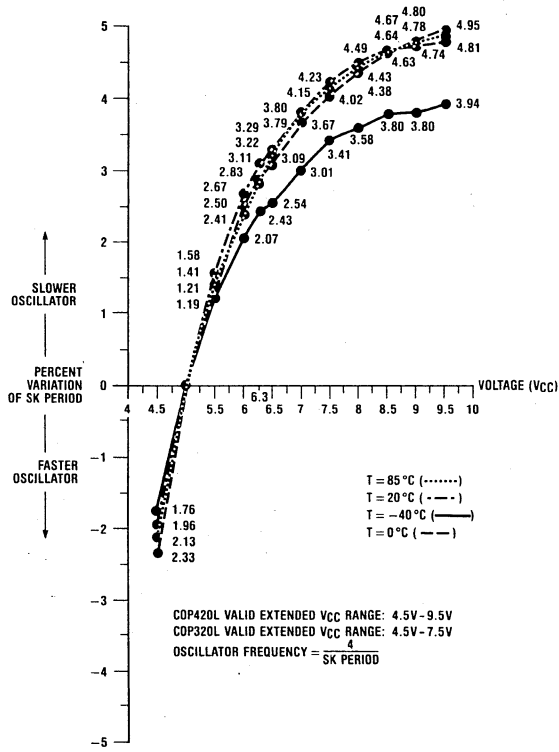
NOTE 1: BASE PERIOD AT V<sub>CC</sub> = 5.0V.  
 NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE RC VARIATION WITH TEMPERATURE.  
 NOTE 3: SK PERIOD = INSTRUCTION CYCLE TIME.

Figure 3. COP320/COP420 RC Oscillator Variation with V<sub>CC</sub>



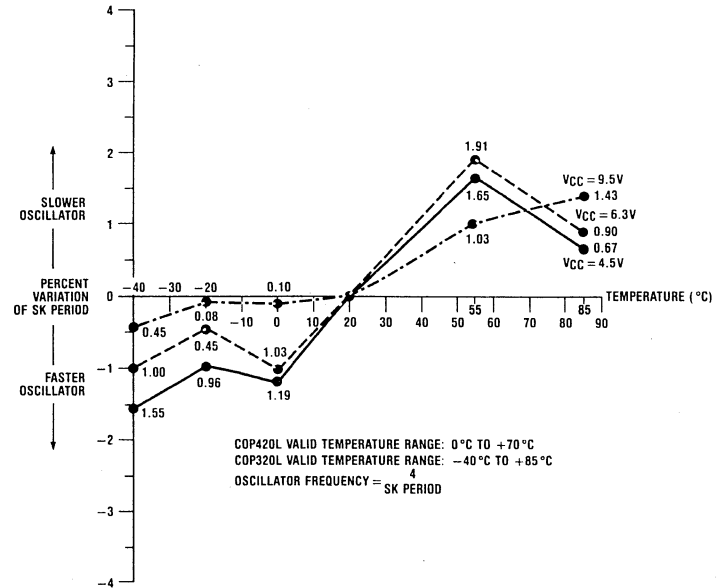
NOTE 1: 20°C = BASE PERIOD.  
 NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE RC VARIATION WITH TEMPERATURE.  
 NOTE 3: SK PERIOD = INSTRUCTION CYCLE TIME.

Figure 4. COP320/COP420 RC Oscillator Variation with Temperature



NOTE 1: BASE PERIOD AT V<sub>CC</sub> = 5.0V.  
 NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE RC VARIATION WITH TEMPERATURE.  
 NOTE 3: SK PERIOD = INSTRUCTION CYCLE TIME.

Figure 5. COP320L/COP420L RC Oscillator Variation with V<sub>CC</sub>



NOTE 1: 20°C = BASE PERIOD.  
 NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE RC VARIATION WITH TEMPERATURE.  
 NOTE 3: SK PERIOD = INSTRUCTION CYCLE TIME.

Figure 6. COP320L/COP420L RC Oscillator Variation with Temperature



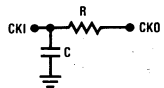


Figure III.1

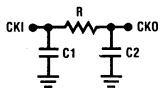


Figure III.2

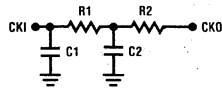


Figure III.3

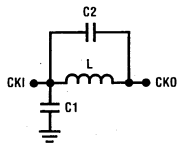


Figure III.4.a

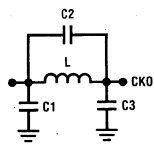


Figure III.4.b

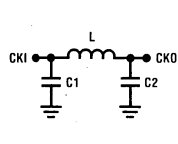


Figure III.4.c

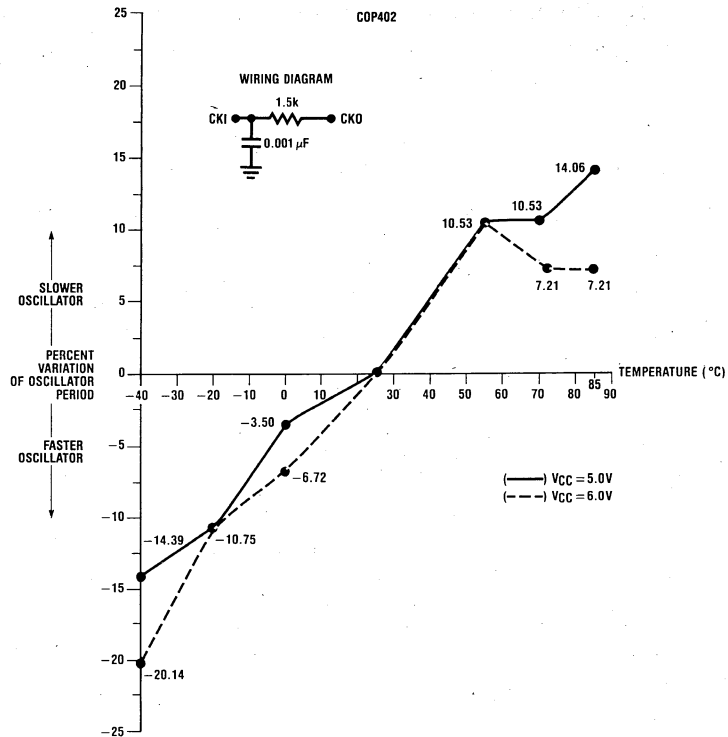
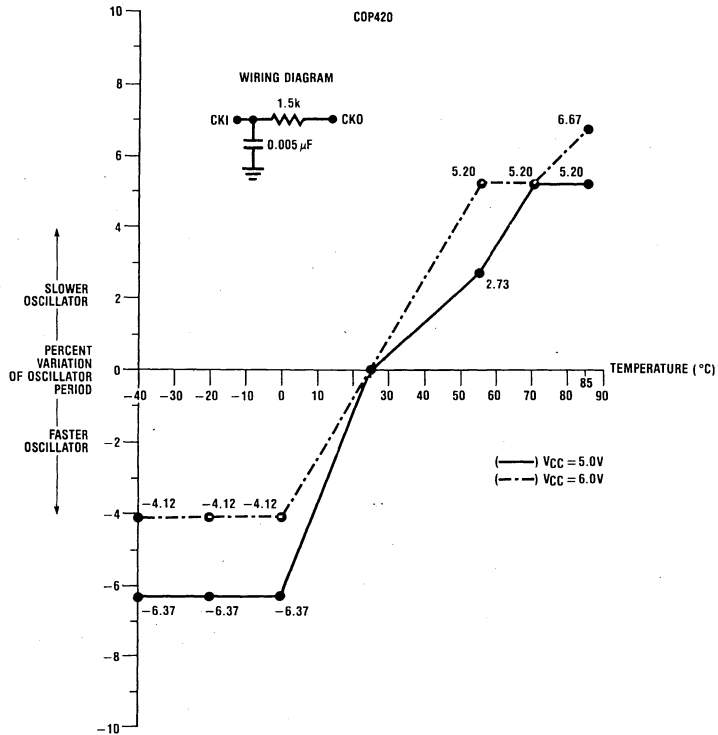
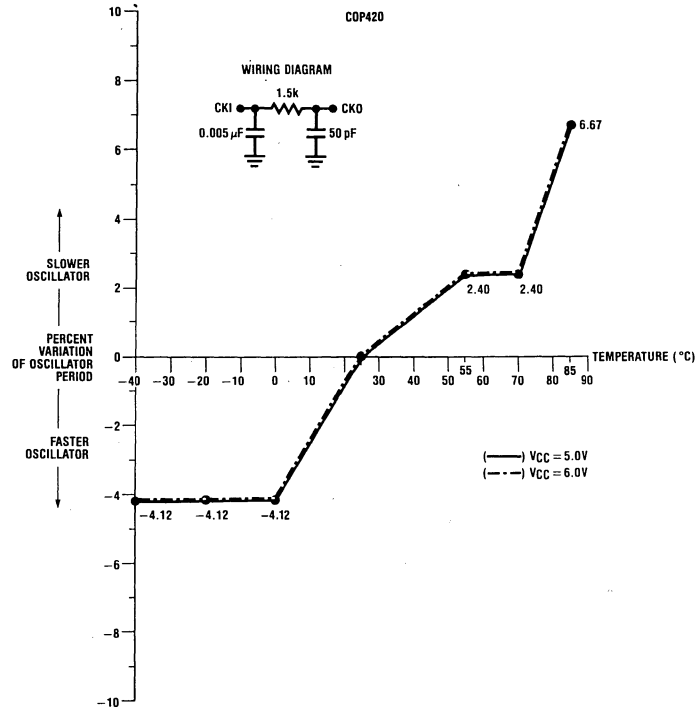


Figure 7



NOTE 1: 25 $^{\circ}$ C = BASE PERIOD.  
NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE RC VARIATION WITH TEMPERATURE.

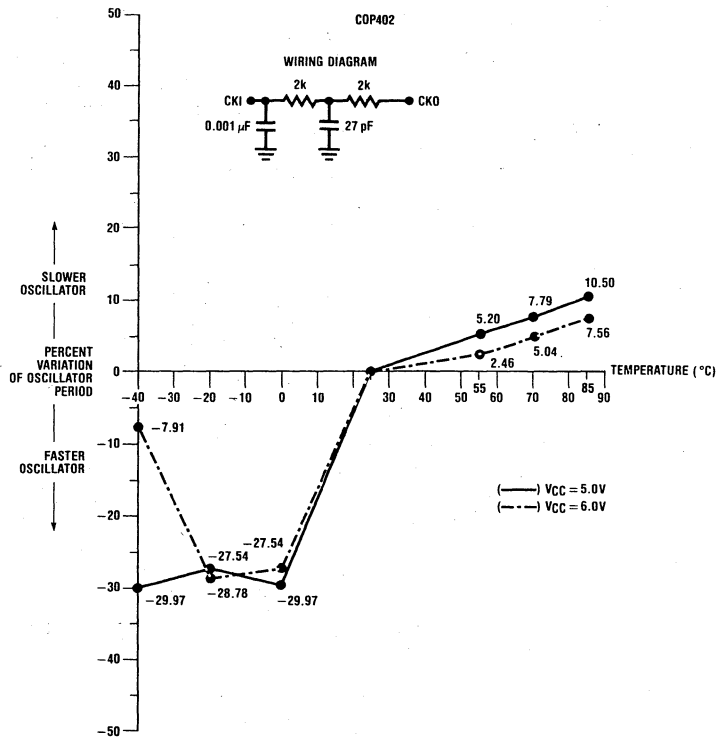
Figure 8



NOTE 1: 25 $^{\circ}$ C = BASE PERIOD.  
NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE RC VARIATION WITH TEMPERATURE.

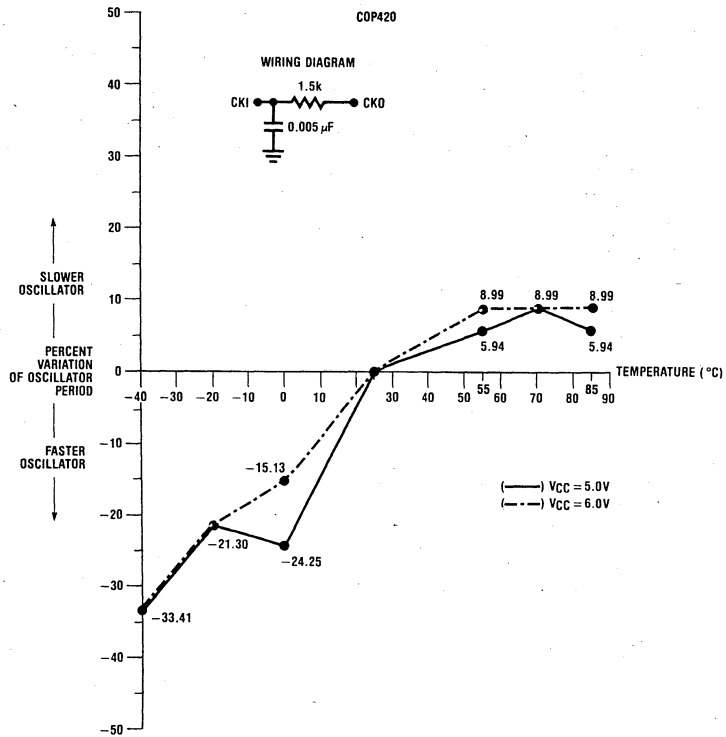
Figure 9





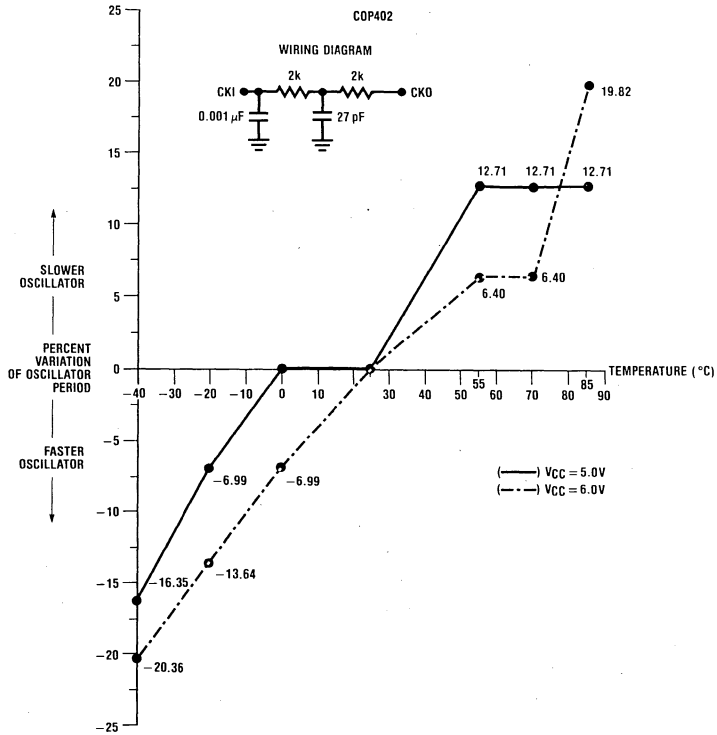
NOTE 1: 25°C = BASE PERIOD.  
 NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE RC VARIATION WITH TEMPERATURE.

Figure 10



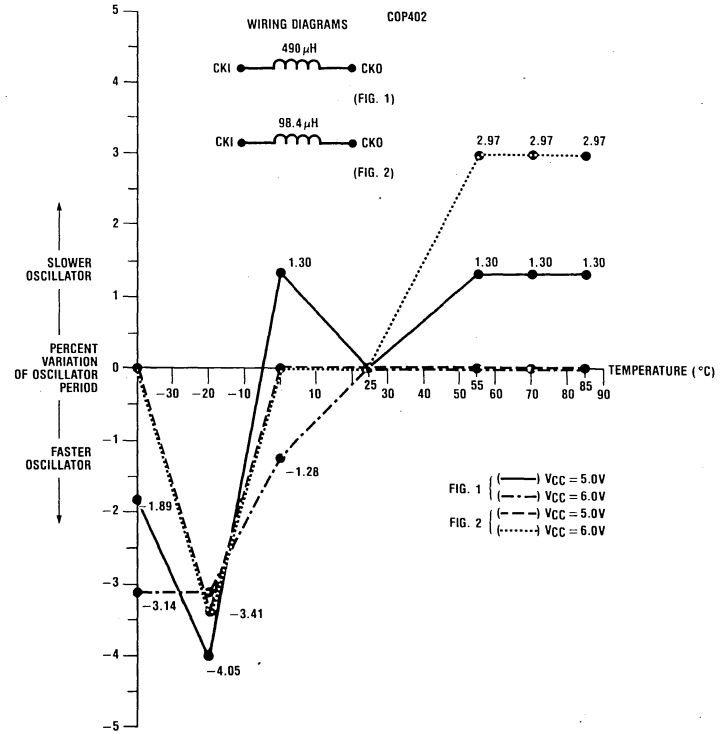
NOTE 1: 25°C = BASE PERIOD.  
 NOTE 2: RC IN OVEN WITH COP420.

Figure 11



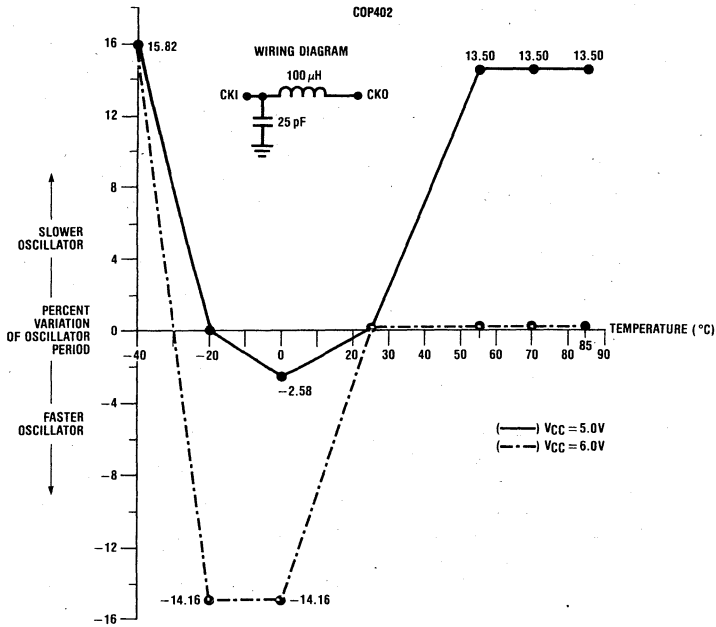
NOTE 1: 25°C = BASE PERIOD.  
 NOTE 2: RC IN OVEN WITH COP402.

Figure 12



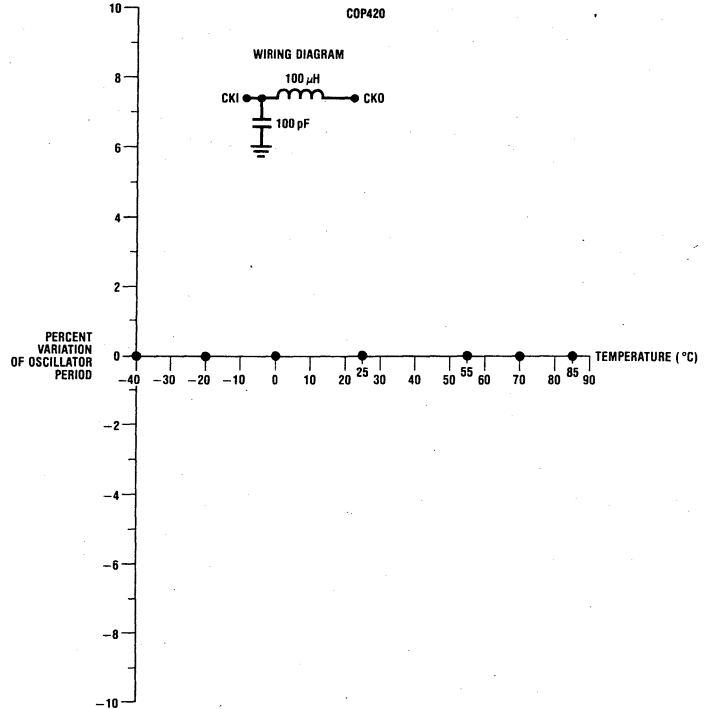
NOTE 1: 25°C = BASE PERIOD.  
 NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE "L" VARIATION WITH TEMPERATURE.

Figure 13



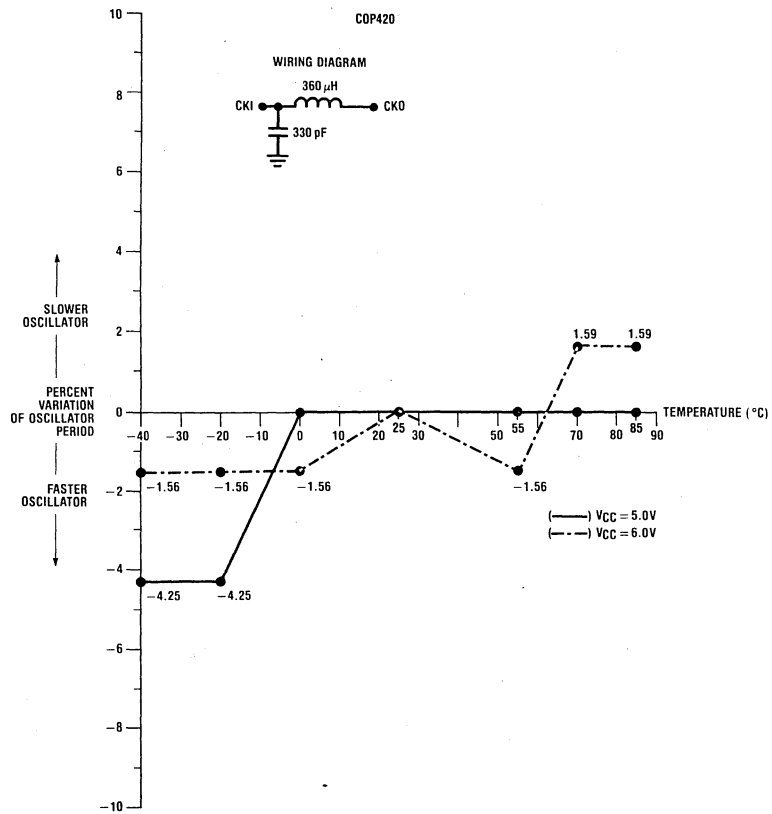
NOTE 1: 25°C = BASE PERIOD.  
 NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE LC VARIATION WITH TEMPERATURE.

Figure 14



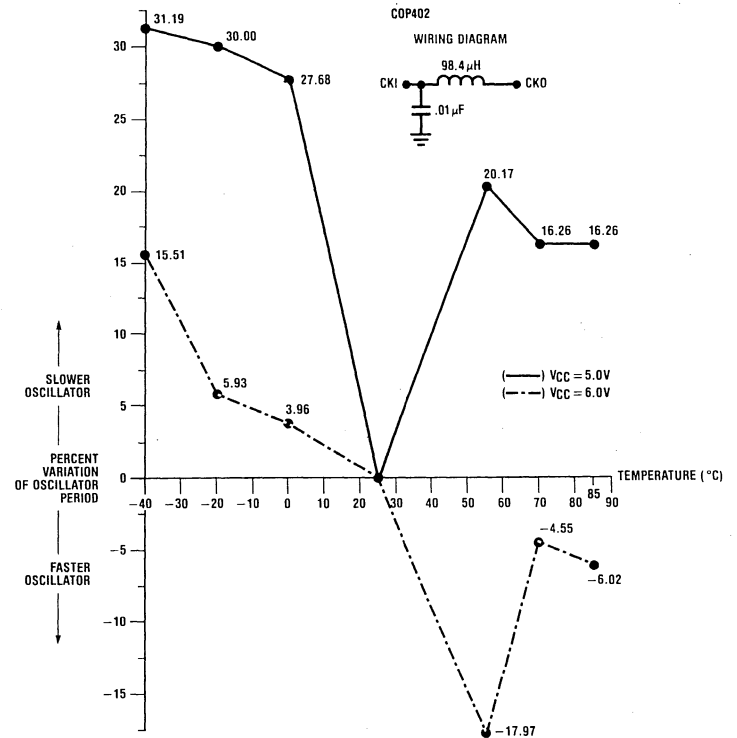
NOTE 1: 25°C = BASE PERIOD.  
 NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE LC VARIATION WITH TEMPERATURE.  
 \*NO MEASURABLE VARIATION OVER TEMPERATURE.

Figure 15



NOTE 1: 25°C = BASE PERIOD.  
 NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE LC VARIATION WITH TEMPERATURE.

Figure 16



NOTE 1: 25°C = BASE PERIOD.  
 NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE LC VARIATION WITH TEMPERATURE.

Figure 17

9-140

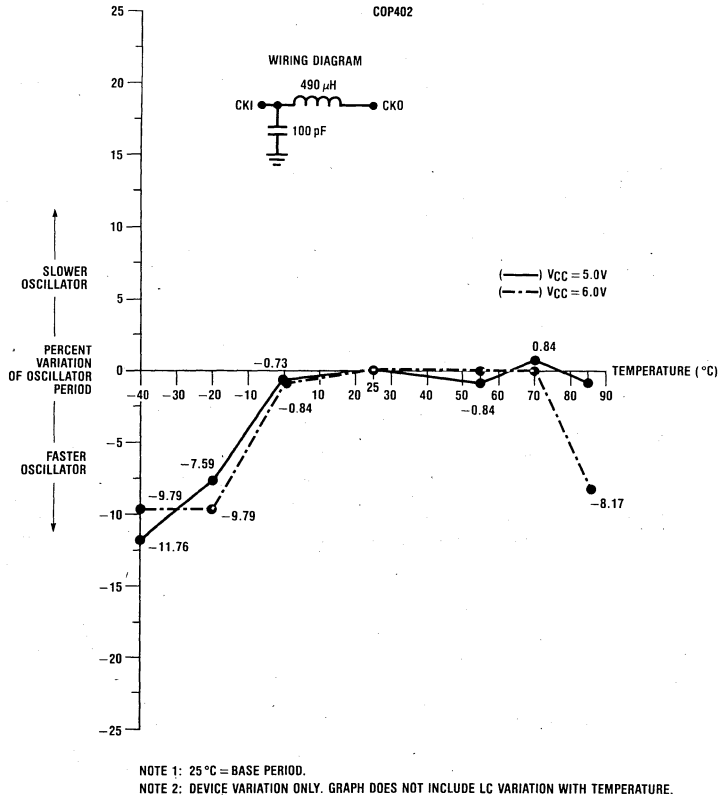


Figure 18

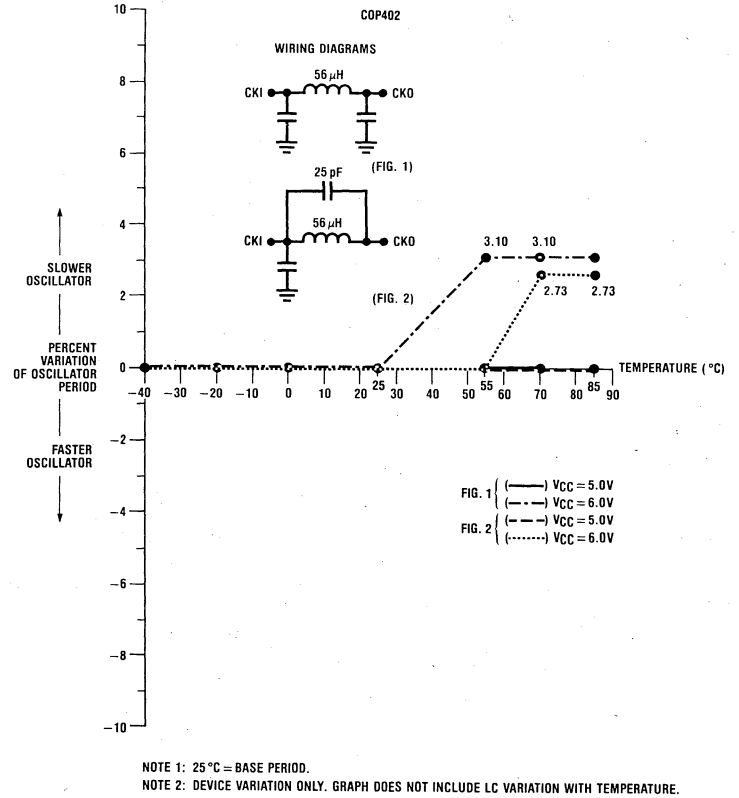
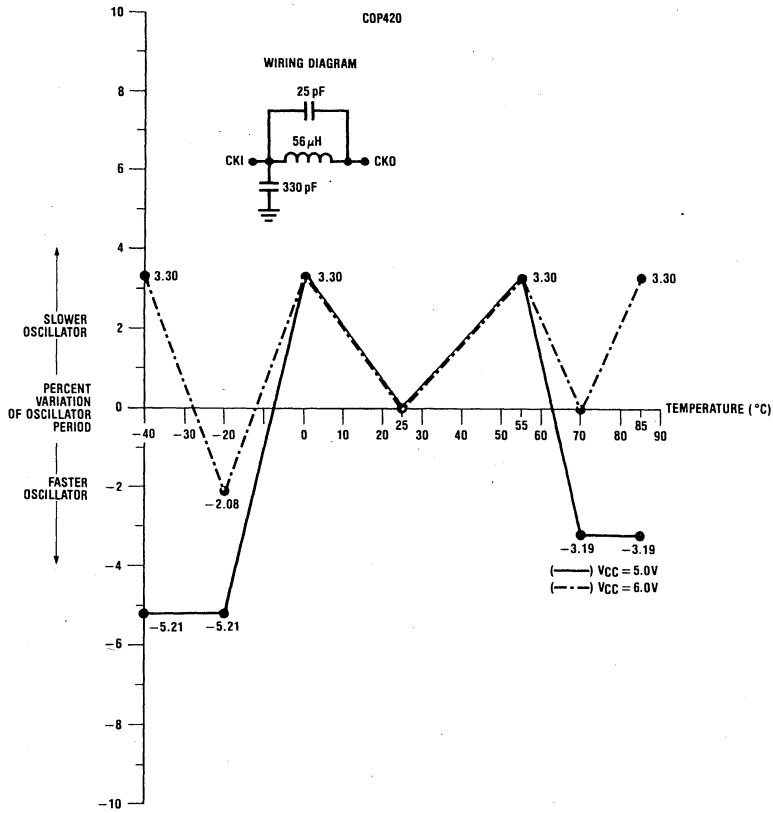
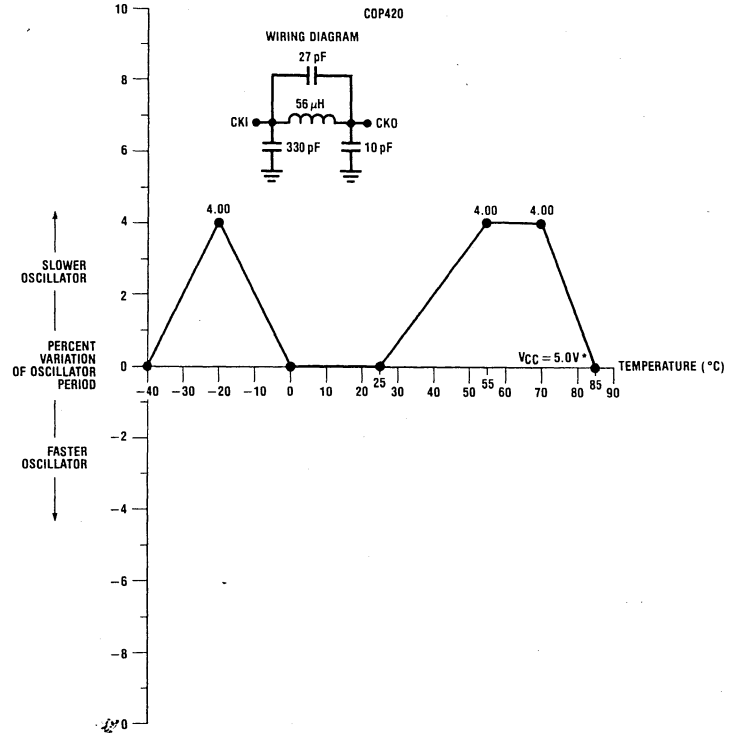


Figure 19



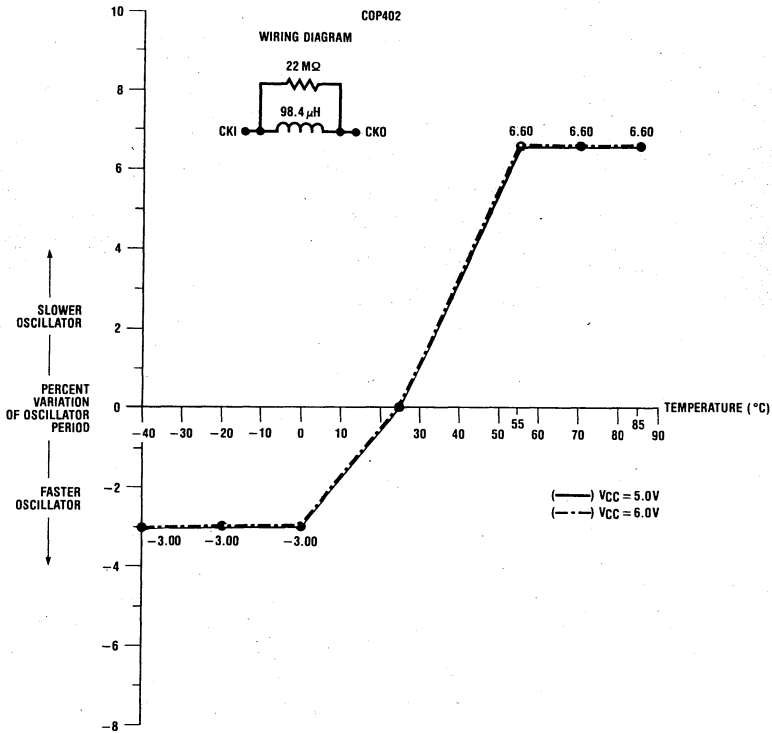
NOTE 1: 25°C = BASE PERIOD.  
NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE LC VARIATION WITH TEMPERATURE.

Figure 20



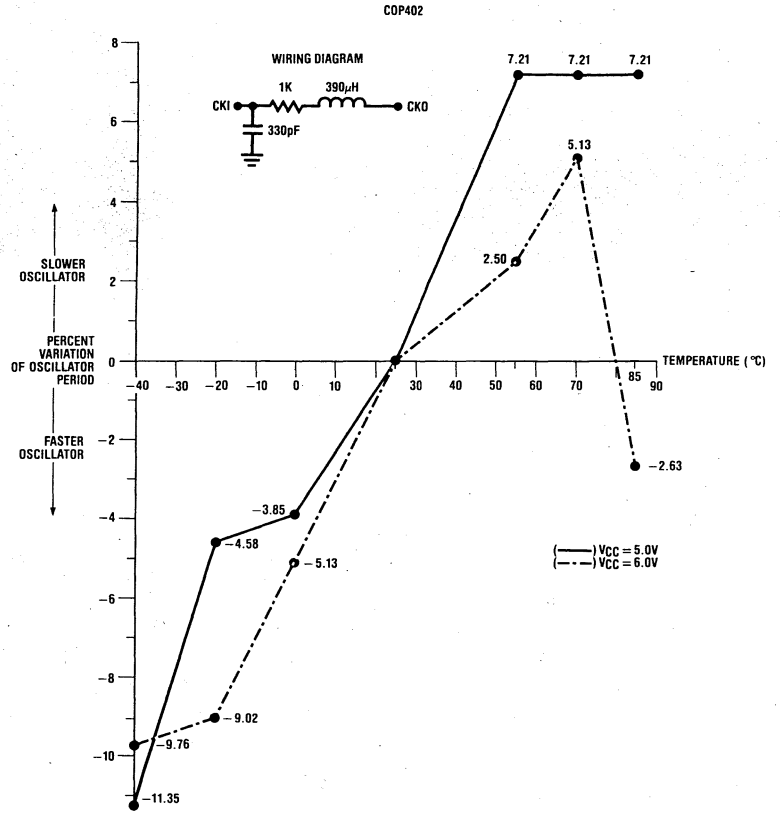
NOTE 1: 25°C = BASE PERIOD.  
NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE LC VARIATION WITH TEMPERATURE.  
\*NO VARIATION AT 6V.

Figure 21



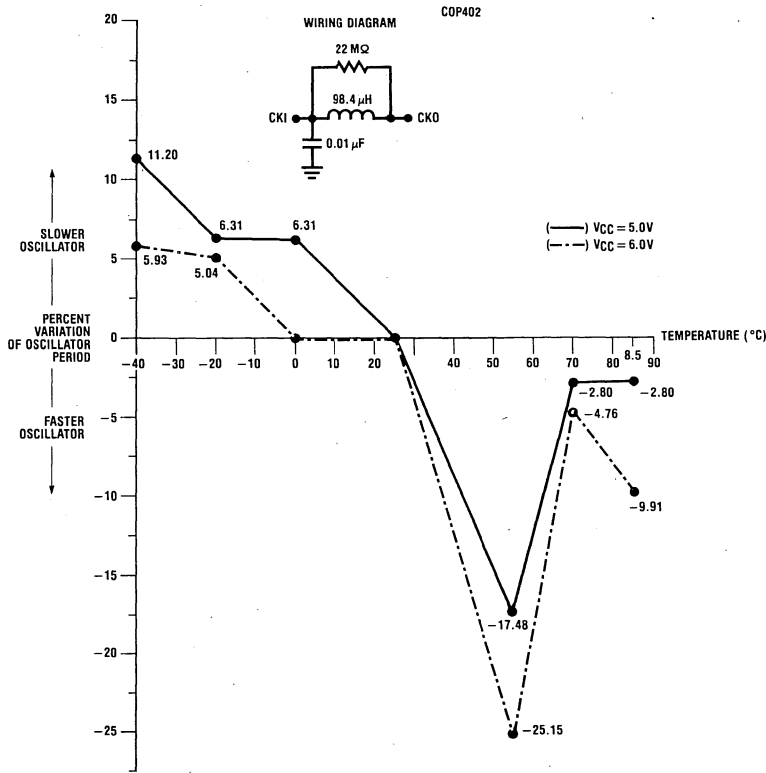
NOTE 1: 25°C = BASE PERIOD.  
NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE RL VARIATION WITH TEMPERATURE.

Figure 22



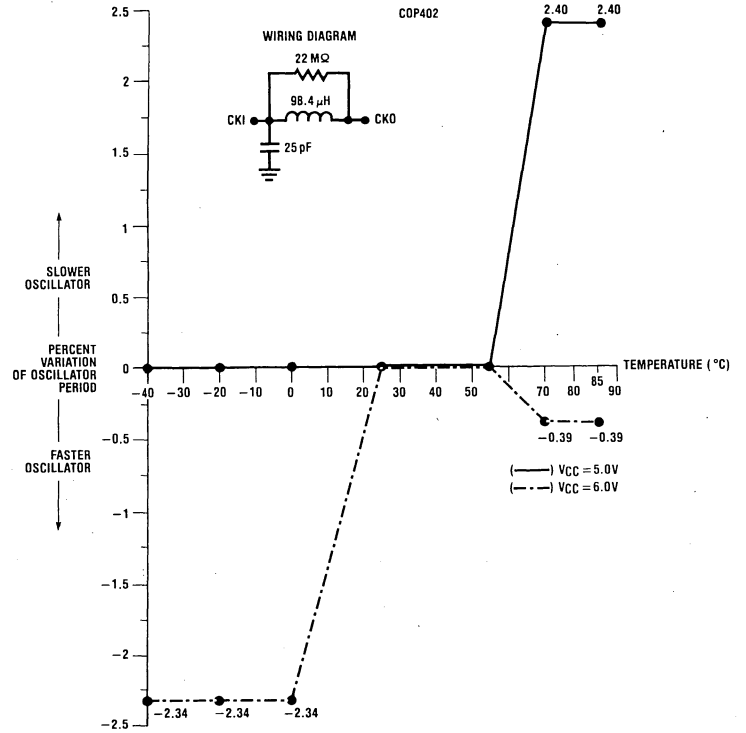
NOTE 1: 25°C = BASE PERIOD.  
NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE RLC VARIATION WITH TEMPERATURE.

Figure 23



NOTE 1: 25°C = BASE PERIOD.  
NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE RLC VARIATION WITH TEMPERATURE.

Figure 24



NOTE 1: 25°C = BASE PERIOD.  
NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE RLC VARIATION WITH TEMPERATURE.

Figure 25



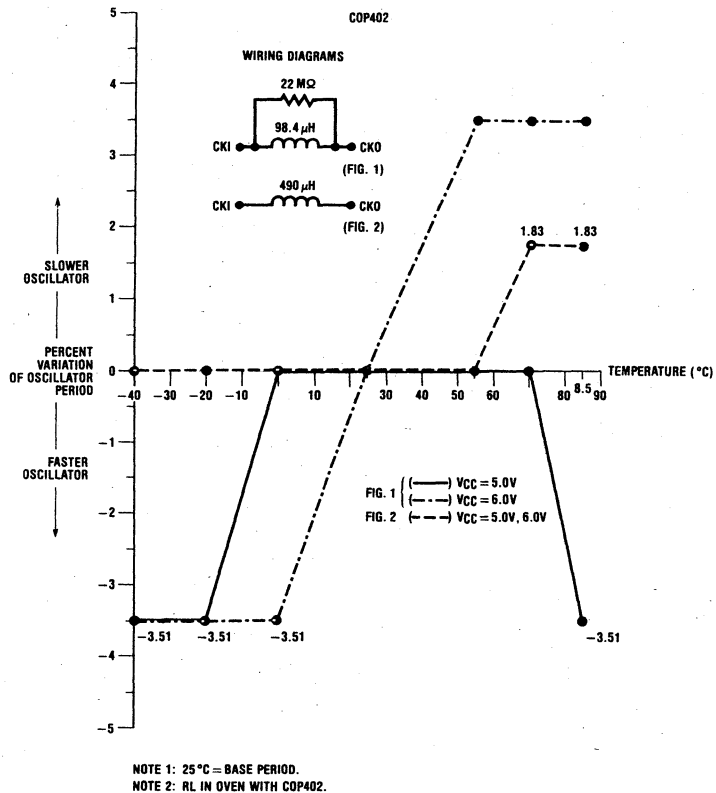


Figure 26

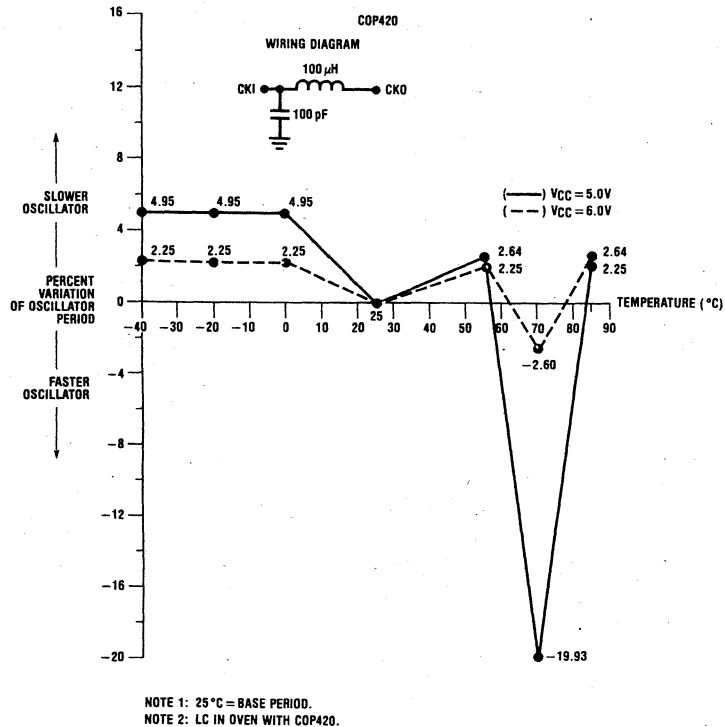
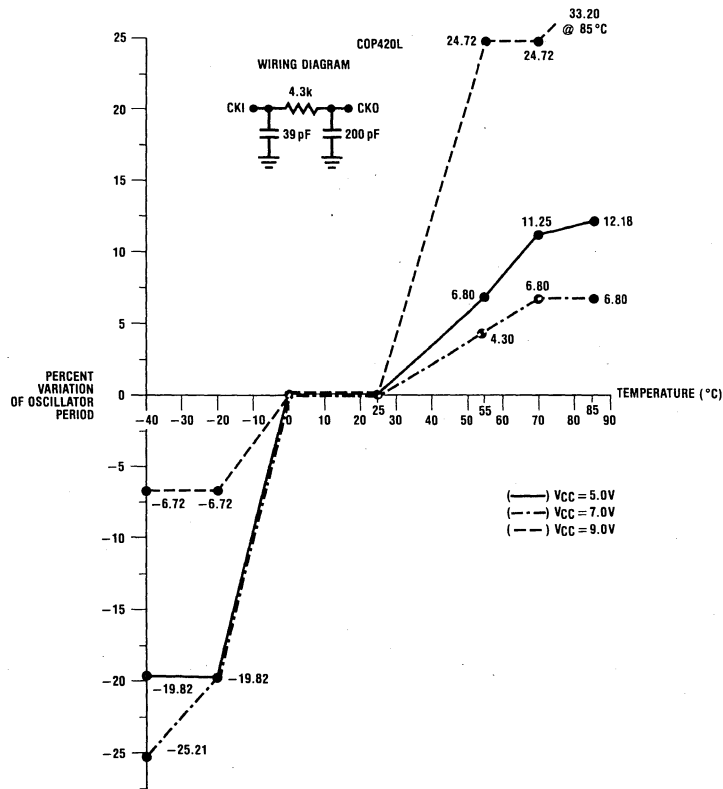
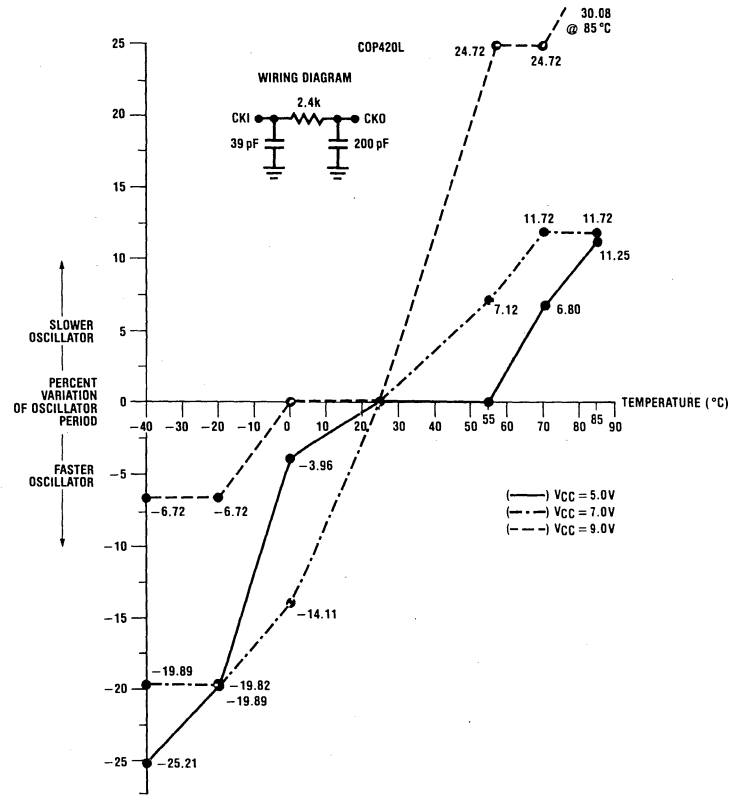


Figure 27



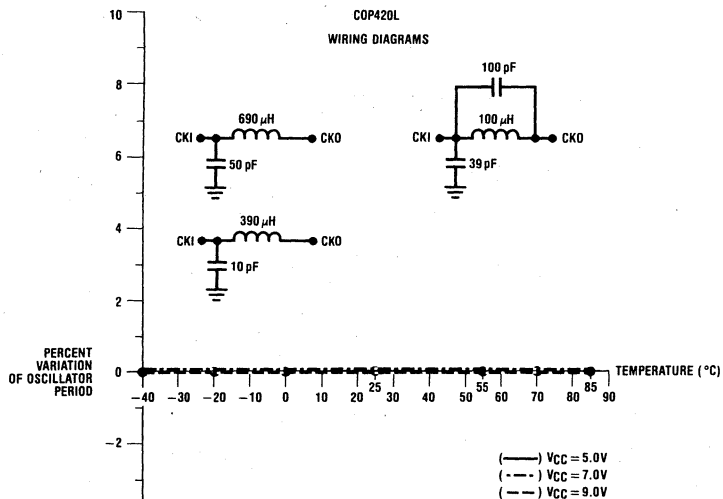
NOTE 1: 25°C = BASE PERIOD.  
NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE RC VARIATION WITH TEMPERATURE.

Figure 28



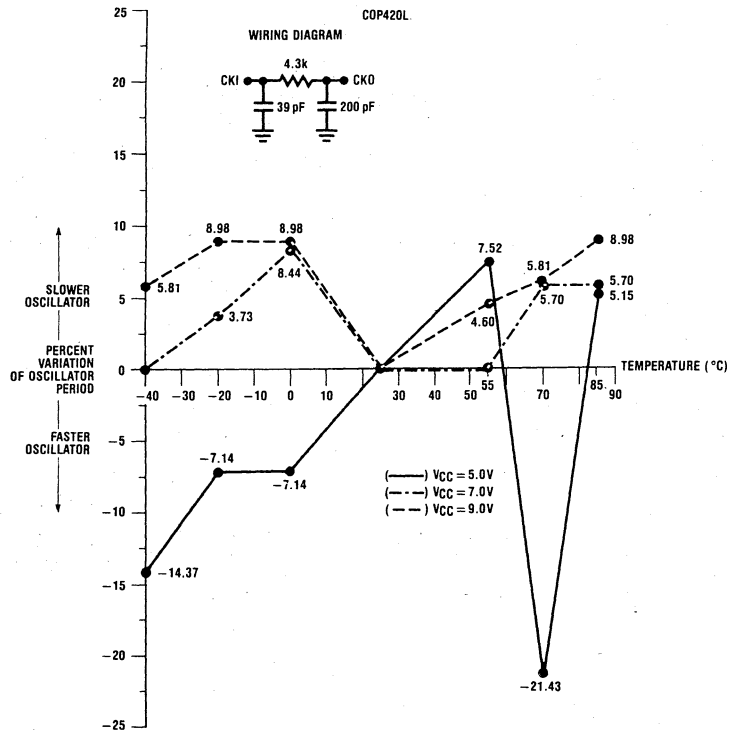
NOTE 1: 25°C = BASE PERIOD.  
NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE RC VARIATION WITH TEMPERATURE.

Figure 29



NOTE 1: NO MEASURABLE VARIATION FOR ALL THREE CIRCUITS ABOVE.  
 NOTE 2: 25°C = BASE PERIOD.  
 NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE LC VARIATION WITH TEMPERATURE.

Figure 30



NOTE 1: 25°C = BASE PERIOD.  
 NOTE 2: RC IN OVEN WITH COP420L.

Figure 31

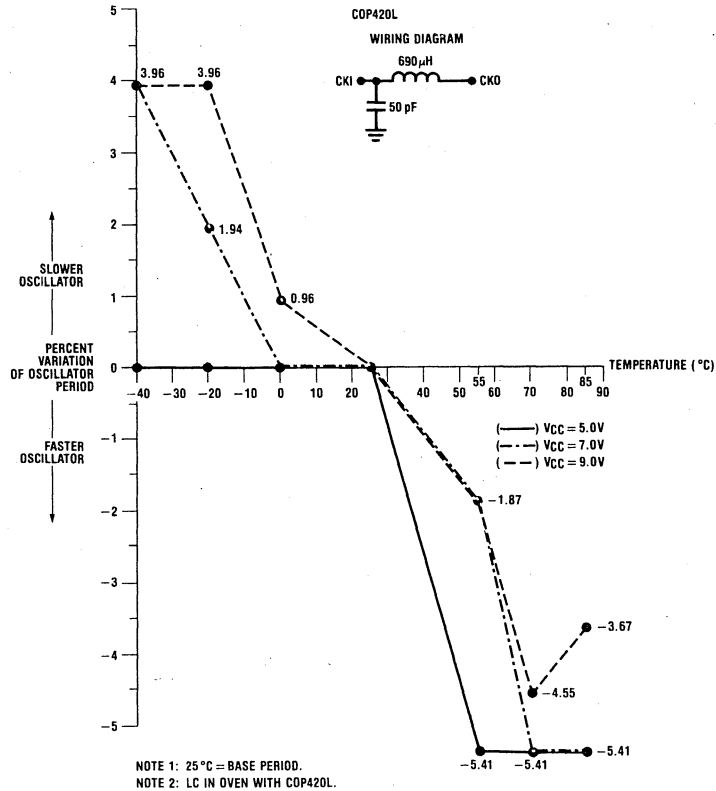


Figure 32

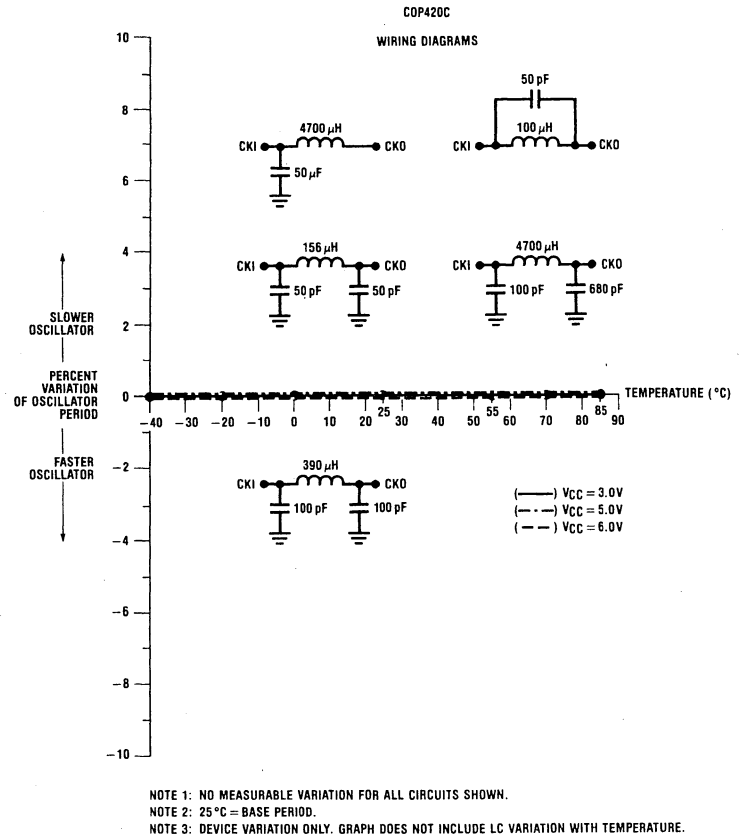


Figure 33

9-148

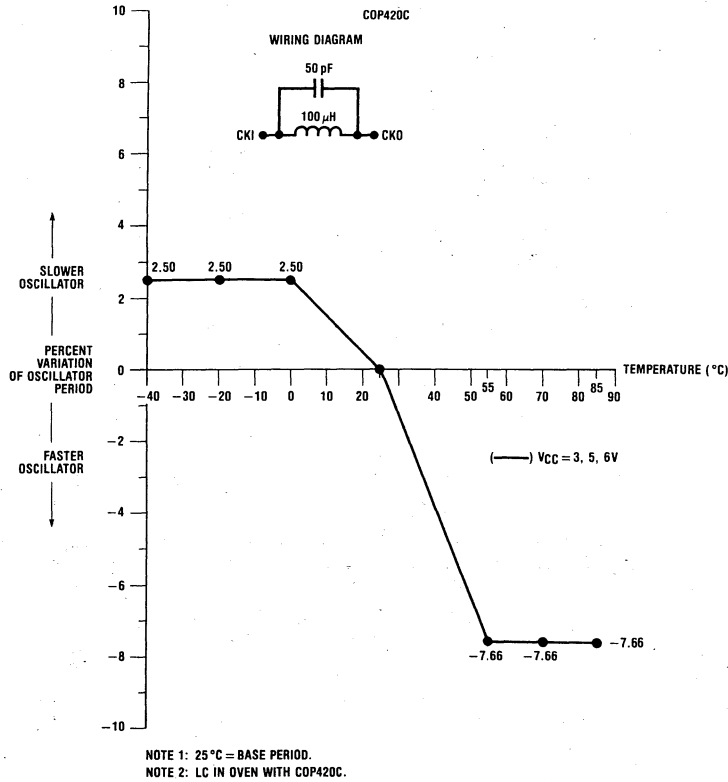


Figure 34

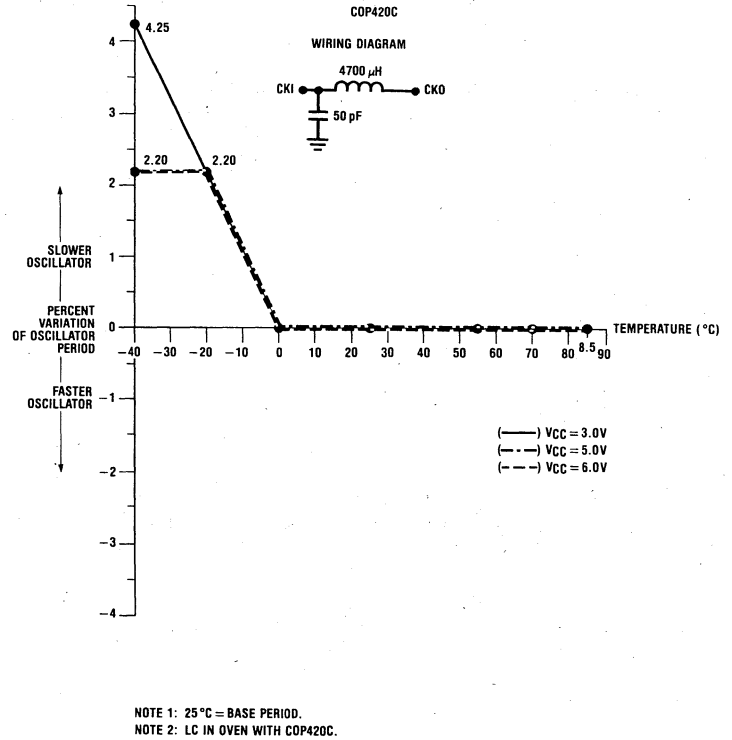
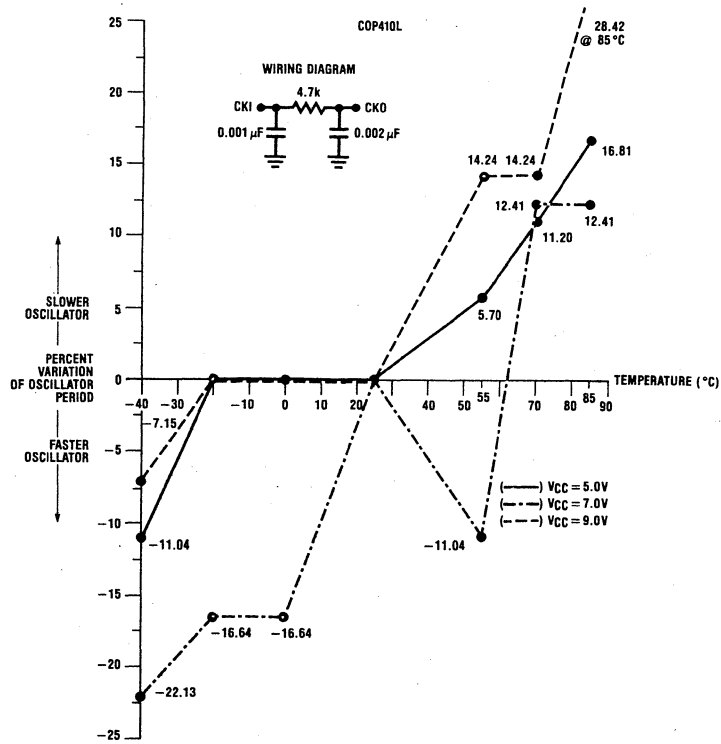
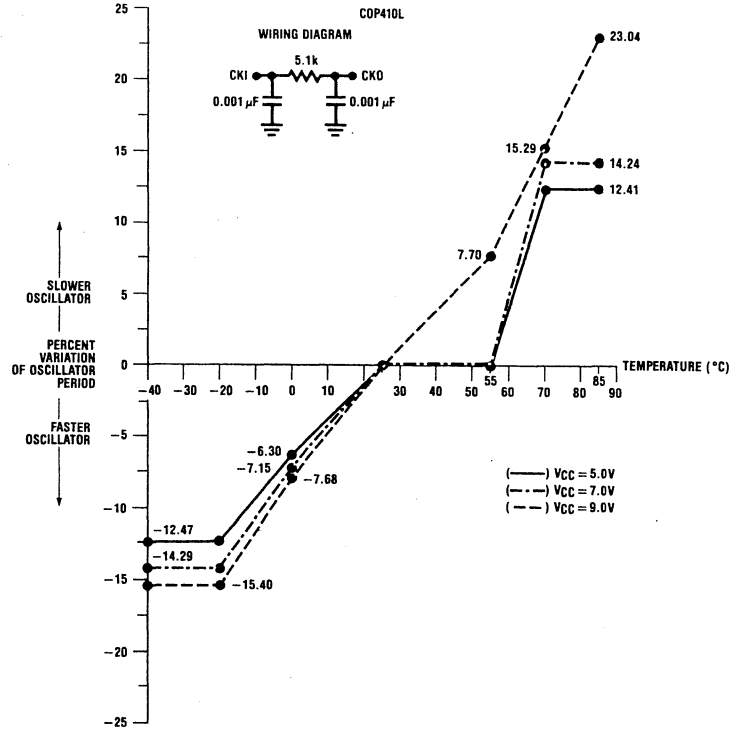


Figure 35



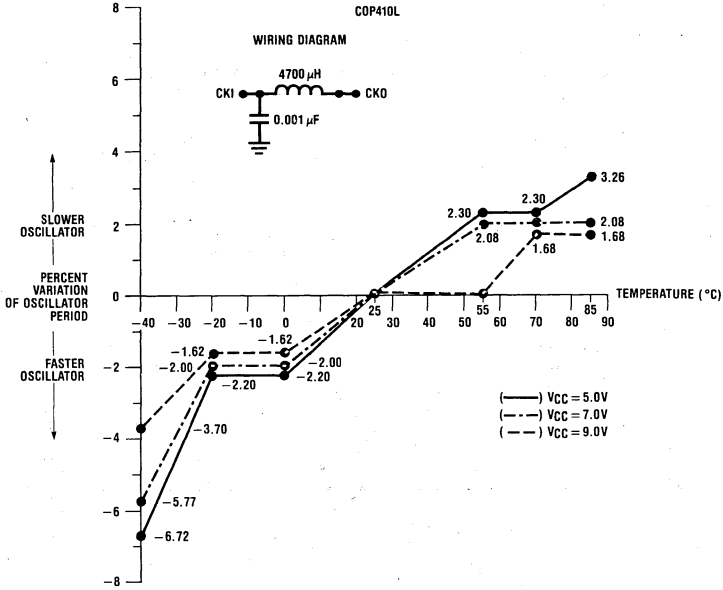
NOTE 1: 25 $^{\circ}$ C = BASE PERIOD.  
NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE RC VARIATION WITH TEMPERATURE.

Figure 36



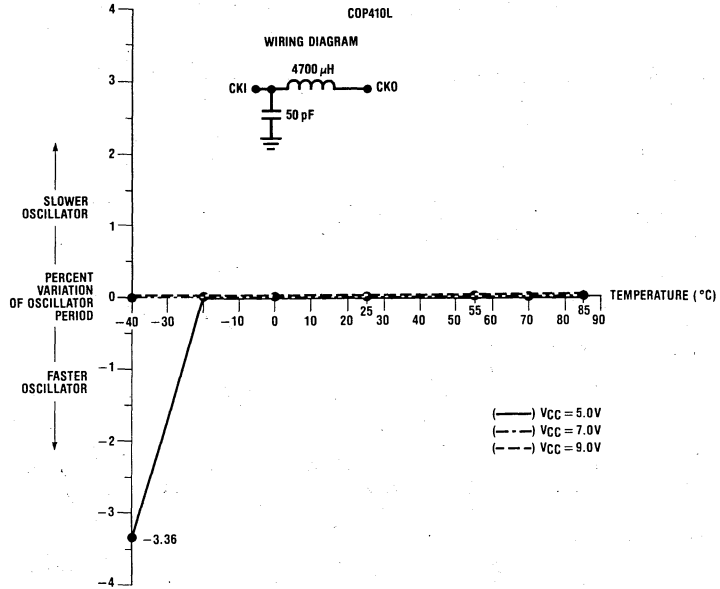
NOTE 1: 25 $^{\circ}$ C = BASE PERIOD.  
NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE RC VARIATION WITH TEMPERATURE.

Figure 37



NOTE 1: 25°C = BASE PERIOD.  
 NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE LC VARIATION WITH TEMPERATURE.

Figure 38



NOTE 1: 25°C = BASE PERIOD.  
 NOTE 2: DEVICE VARIATION ONLY. GRAPH DOES NOT INCLUDE LC VARIATION WITH TEMPERATURE.

Figure 39

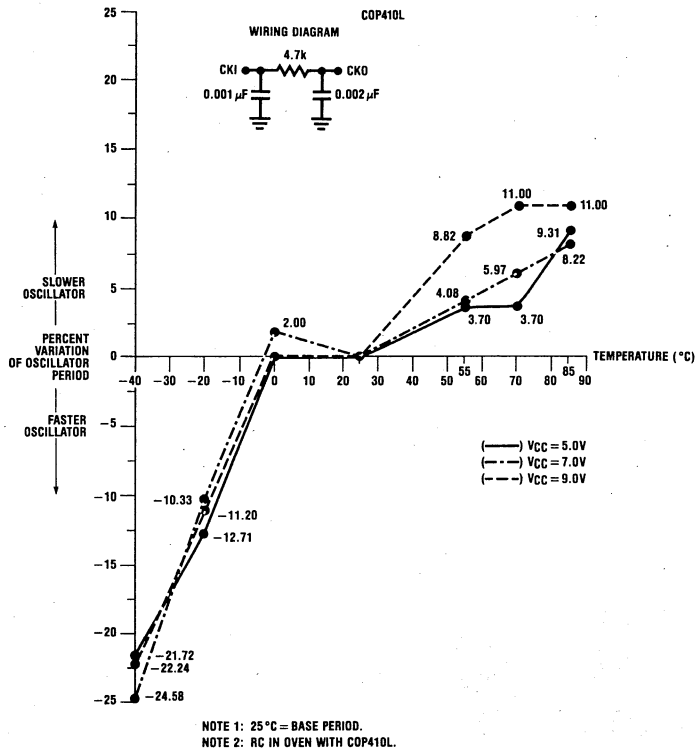


Figure 40

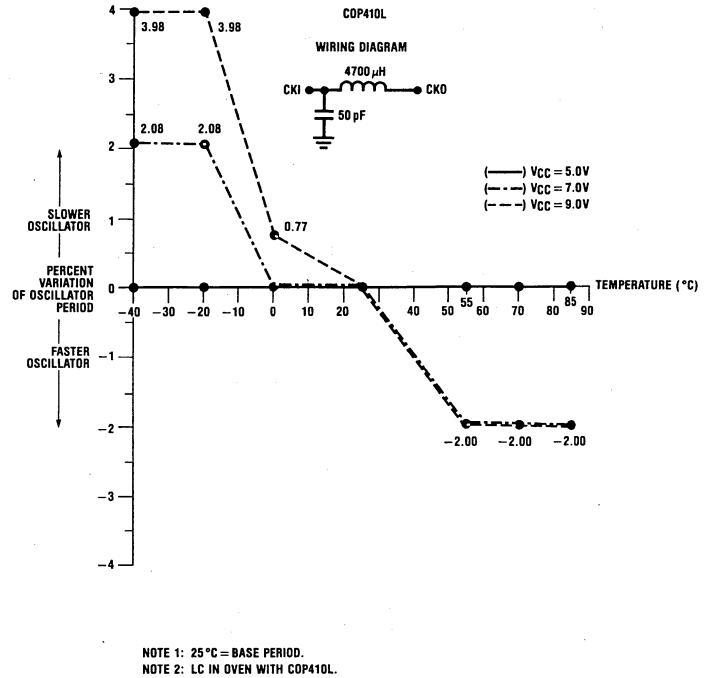


Figure 41



# Triac Control Using the COP400 Microcontroller Family

National Semiconductor  
COP Note 6  
February 1981



## Table of Contents

I. Triac Control .....	9-152
A. Basic Triac Operation .....	9-152
B. Triggering .....	9-153
C. Zero Voltage Detection .....	9-153
D. Direct Couple .....	9-153
E. Pulse Transformer Interface .....	9-154
F. False Turn-on .....	9-154
II. Software Techniques .....	9-154
A. Zero Voltage Detection .....	9-154
B. Processing Time Allocations .....	9-155
1. Half Cycle Approach .....	9-155
2. Full Cycle Approach .....	9-155
C. Steady State Triggering .....	9-156
III. Triac Light Intensity Control Code .....	9-157
A. Triac Light Intensify Routine .....	9-157

## Triac Control

The COP400 single-chip controller family members provide computational ability and speed which is more than adequate to intelligently manage power control. These controllers provide digital control while low cost and short turnaround enhance COPSTM desirability. The COPS controllers are capable of 4  $\mu$ s cycle times which can provide more than adequate computational ability when controlling 60Hz line voltage. Input and output options available on the COPS devices can contour the device to apply in many electrical situations. A more detailed description of COPS qualifications is available in the COP400 data sheets.

The COPS controller family may be utilized to manage power in many ways. This paper is devoted to the investigation of low cost triac interfaces with the COP400 family microcontroller and software techniques for power control applications.

## BASIC TRIAC OPERATION

A triac is basically a bidirectional switch which can be used to control AC power. In the high-impedance state, the triac blocks the principal voltage across the main terminals. By pulsing the gate or applying a steady state gate signal, the triac may be triggered into a low impedance state where conduction across the main terminals will occur. The gate signal polarity need not follow the main terminal polarity; however, this does affect the gate current requirements. Gate current requirements vary depending on the direction of the main terminal current and the gate current. The four trigger modes are illustrated in Figure 1.

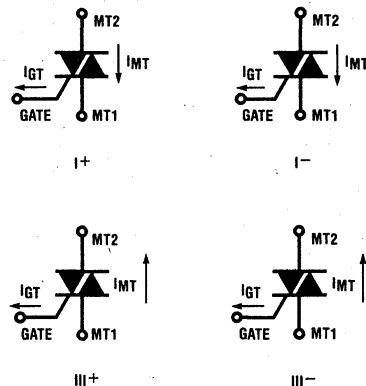


Figure 1. Gate Trigger Modes. Polarities Referenced to Main Terminal 1.

The breakover voltage ( $V_{BO}$ ) is specified with the gate current ( $I_{GT}$ ) equal to zero. By increasing the gate current supplied to the triac,  $V_{BO}$  can be reduced to cause the triac to go into the conduction or on state. Once the triac has entered the on state the gate signal need not be present to sustain conduction. The triac will turn itself off when the main terminal current falls below the minimum holding current required to sustain conduction ( $I_H$ ).

A typical current and voltage characteristic curve is given in Figure 2. As can be seen, when the gate voltage and the main terminal 2 (MT2) voltages are positive with respect to MT1 the triac will operate in quadrant 1. In this case the trigger circuit sources current to the triac (I+ MODE).

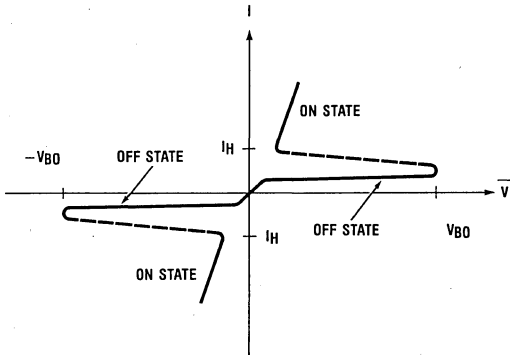


Figure 2. Voltage-Current Characteristics

After conduction occurs the main terminal current is independent of the gate current; however, due to the structure of the triac the gate trigger current is dependent on the direction of the main terminal current. The gate current requirements vary from mode to mode. In general, a triac is more easily triggered when the gate current is in the same direction as the main terminal current. This can be illustrated in the situation where there is not sufficient gate drive to cause conduction when MT2 is both positive and negative. In this case the triac may act as a single direction SCR and conduction occurs in only one direction. The trigger circuit must be designed to provide trigger currents for the worst case trigger situation. Another reason ample trigger current must be supplied is to prevent localized heating within the pellet and speed up turn-on time. If the triac is barely triggered only a small portion of the junction will begin to conduct, thus causing localized heating and slower turn-on. If an insufficient gate pulse is applied damage to the triac may result.

## TRIGGERING

Gate triggering signals should exceed the minimum rated trigger requirements as specified by the manufacturer. This is essential to guarantee rapid turn-on time and consistent operation from device to device.

Triac turn-on time is primarily dependent on the magnitude of the applied gate signal. To obtain decreased turn-on times a sufficiently large gate signal should be applied. Faster turn-on time eliminates localized heat spots within the pellet structure and increases triac dependability.

Digital logic circuits, without large buffers, may not have the drive capabilities to efficiently turn on a triac. To insure proper operation in all firing situations, external trigger circuitry might become necessary. Also, to prevent noise from disturbing the logic levels, AC/DC isolation or coupling techniques must be utilized. Sensitive gate triacs which require minimal gate input signal and provide a limited amount of main terminal current may be driven directly. This paper will focus on 120 V<sub>AC</sub> applications of power control.

## ZERO VOLTAGE DETECTION

In many applications it is advantageous to switch power at the AC line zero voltage crossing. In doing this, the device being controlled is not subjected to inherent AC transients. By utilizing this technique, greater dependability can be obtained from the switching device and the device being switched. It is also sometimes desirable to reference an event on a cyclic basis corresponding to the AC line frequency. Depending on the load characteristics, switching times need to be chosen carefully to insure optimal performance. Triac controlled AC switching referenced to the AC 60Hz line frequency enables precise control over the conduction angle at which the triac is fired. This enables the COPS device to control the power output by increasing or decreasing the conduction angle in each half cycle.

A wide variety of zero voltage detection circuits are available in various levels of sophistication. COPS devices, in most cases, can compensate for noisy or semi-accurate ZVD circuits. This compensation is utilized in the form of debounce and delay routines. If a noisy transition occurs near zero volts the COPS device can wait for a valid transition period specified by the maximum amount of noise present. Some software considerations are presented in the software section and are commented upon. The minimal detection circuit is shown in Figure 9.

## DIRECT COUPLE

Isolation associated problems can be overcome by means of direct AC coupling. One such method is illustrated in Figure 3. This circuit incorporates a half-wave rectifier in conjunction with a filter capacitor to provide the logic power supply. The positive half-cycle is allowed to drop across the zener diode and be filtered by the capacitor. This creates a low cost line interface; however, only a limited supply current is available. In order to control the current capabilities of this circuit the series resistor must be modified. However, as more current is required, the power that must be dissipated in the series resistor increases. This increases the power dissipation requirements of the series resistor and the system cost. For applications which require large current sources an alternative method is advisable. In order to assure consistent operation, power supply

ripple must be minimized. COPS devices can be operated over a relatively wide power supply range. However, excessive ripple may cause an inadvertent reset operation of the device.

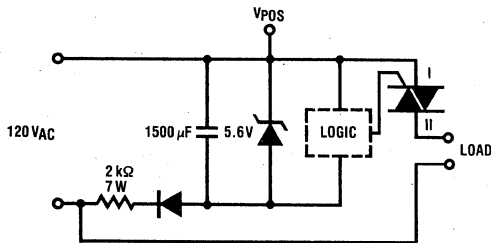


Figure 3. AC Direct Couple

### PULSE TRANSFORMER INTERFACE

Digital logic control of triacs is easily accomplished by triggering through pulse transformers or optical coupling. The energy step-up gained by using a pulse transformer should provide a more than adequate gate trigger signal. This complies with manufacturers' suggested gate signal requirements. Pulse transformers also provide AC/DC isolation necessary in control logic interfaces. Minimal circuit interface to the pulse transformer is required as shown in Figure 4. Optical coupling circuits provide isolation, and in some cases adequate gate drive capabilities.

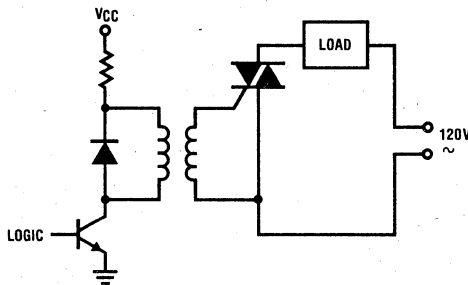


Figure 4. Pulse Transformer Interface

A logic controlled pulse is applied to the base of the transistor to switch current through the primary of the pulse transformer. The transformer then transfers the signal to the secondary and causes the triac to fire. The energy transfer that is now available on the secondary is more than adequate to turn on the triac in any of its operating modes. When the pulse transformer is switched off a reverse EMF is generated in the primary coil which may cause damage to the transistor. The diode across the primary serves to protect the collector junction of the switching transistor. Another major advantage is AC isolation; the gate of the triac is now completely isolated from the logic portion of the circuit.

### FALSE TURN-ON

When switching an inductive load, voltage spikes may be generated across the main terminals of the triac which have the potential of a non-gated turn-on of the triac. This creates the undesirable situation of limited control of the system. In a system with an inductive load the voltage leads the current by a phase shift corresponding to the amount of inductance in the motor. As the current passes near zero, the voltage is at a non-zero value, offset due to the phase shift. When the principal current through the triac pellet decreases to a value not capable of sustaining conduction the triac will turn off. At this point in time the voltage across the terminals will instantaneously attain a value corresponding to the phase shift caused by the inductive load. The rapid decay of current in the inductor causes an  $L di/dt$  voltage applied across the terminals of the triac. Should this voltage exceed the blocking voltage specified for the triac, a false turn-on will occur.

In order to avoid false turn-on, a snubber network must be added across the terminals to absorb the excess energy generated by this situation. A common form of this network is a simple RC in series across the terminals. In order to select the values of the network it is necessary to determine the peak voltage allowable in the system and the maximum  $dV/dt$  stress the triac can withstand. One approach to obtaining the optimal values for  $R_S$  and  $C_S$  is to model the effective circuit and solve for the triac voltage. The snubber in conjunction with the load can now be modeled as an RLC network. Due to the two storage elements (L motor, C snubber) a second order differential equation is generated. Rather than approach this problem from a computer standpoint it becomes much easier to obtain design curves generated for rapid solution of this problem. These design curves are available in many triac publications. (For instance, see RCA application note AN 4745.)

### Software Techniques

#### ZERO VOLTAGE DETECTION

In order to intelligently control triacs on a cyclic basis, an accurate time base must be defined. This may be in the form of an AC, 60 Hz sync pulse generated by a zero voltage detection circuit or a simple real time clock. The COP400 series microcontrollers are suited to accommodate either of these time base schemes while accomplishing auxiliary tasks.

Zero voltage detection is the most useful scheme in AC power control because it affords a real time clock base as well as a reference point in the AC waveform. With this information it is possible to minimize RFI by initiating power-on operations near the AC line voltage zero crossing. It is also possible to fire the triac for only a portion of the cycle, thus utilizing conduction angle manipulation. This is useful in both motor control and light intensity control.

Sophisticated zero voltage detection circuits which are capable of discriminating against noise and switch precisely at zero crossing are not necessary when used in conjunction with a COPS device. COPS software is capable of compensating for noisy or semi-accurate zero voltage detection circuits. This can be accomplished by introducing delays and debounce techniques in the software routines. With a given reference point in the AC

waveform it now becomes easy to divide the waveform to efficiently allocate processing time. These techniques are illustrated in the code listing at the end of this paper.

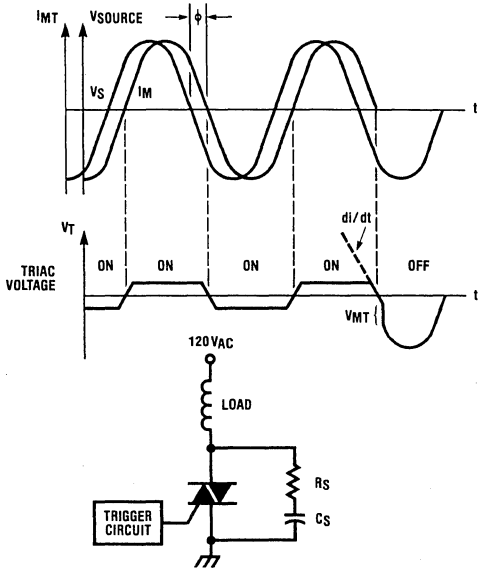


Figure 5. Current Lag Caused by Inductive Load, Snubber Circuit

**PROCESSING TIME ALLOCATIONS**

**Half Cycle Approach**

In order to accomplish more than triac timing, dead delay time must be turned into computation time. It appears that the controller is occupied totally by time delays, which leaves a very limited amount of additional control capability. There are, however, many ways to accomplish auxiliary tasks simultaneously.

On each half cycle an initial delay is incorporated to space into the cycle. This dead time may be put to use and very little voltage to the load is sacrificed. For example, if the load is switched on at  $\pi/4$  RAD, the maximum applied RMS voltage to the load is  $114V_{RMS}$  (assuming  $V_{SUPPLY} = 120V_{RMS}$ ). This is illustrated in the figure below.

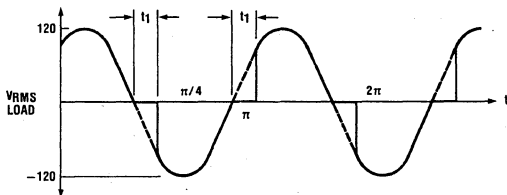


Figure 6. Full Cycle Approach

If a delay of  $\pi/4$  RAD (45 degrees) is inserted after each zero crossing detection the RMS voltage to the load can be determined in the following manner:

$$V_{LOAD} = \sqrt{\frac{(120\sqrt{2})^2}{(2)\pi} (2) \int_{\pi/4}^{\pi} \sin^2(a) da}$$

$$V_{LOAD} = \sqrt{\frac{(120\sqrt{2})^2}{(2)\pi} (2) (1.428)}$$

$$V_{LOAD} = 114.4 V_{RMS}$$

$$\pi/4 \text{ RAD} = 45 \text{ degrees} \quad @ \text{ 60Hz} \quad t = 2.08 \text{ ms}$$

As can be seen the dead time on each half cycle can be 2.08 ms and the load will still see  $114.4V_{RMS}$  of a  $V_{SUPPLY}$  of  $120V_{RMS}$ . If this approach is implemented the initial delay of 2.08ms can be used as computation time. The number of instructions which can be executed when operating at  $4\mu s$  instruction cycle time is:

$$2.08 \text{ ms} / 4\mu s = 520 \text{ instructions}$$

(130 instructions at  $16\mu s$  cycle time)

**Full Cycle Approach**

The methods of half cycle and full cycle triggering are very similar in procedure. The main difference is that all timing is referenced from only one (of the two) zero voltage detection transition in each full AC cycle. For most all applications, when varying the conduction angle it is desirable to fire at the same conduction angle each half cycle to maintain a symmetric applied voltage. In order to accomplish this the triac may be fired twice from one reference point. When applying this technique an 8.33ms delay must be executed to maintain the symmetric applied voltage. This approach provides the most auxiliary computation time in that the 8.33ms delay may be turned into computational time. The basic flow for this technique is illustrated below.

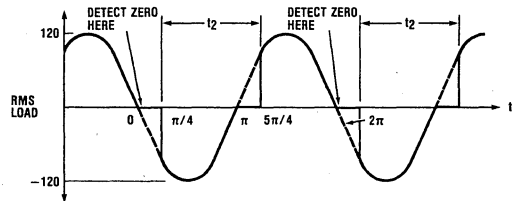


Figure 7. Full Cycle Approach

In the above example the zero crossing pulse is de-bounced on the one-to-zero transition, thus marking the beginning of a full cycle. Once this transition has been

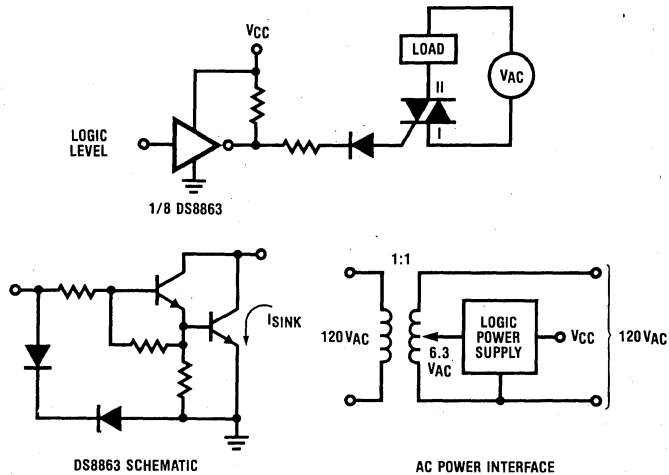


Figure 8. Steady State Triggering

detected an initial delay of  $\pi/4$  RAD is incorporated and the triac is fired. At this time exactly 8.33 ms is available until the triac need be triggered again. This will provide a symmetric voltage to the load only if the delay is 8.33 ms. During this period the number of instructions which can be executed when operating at  $4\mu\text{s}$  is:

$$\frac{8.33 \text{ ms}}{4 \mu\text{s}} = 2082$$

(520 instructions at  $16\mu\text{s}$ )

An alternative approach may be to take the burden from the COPS device by using peripheral devices such as static display controllers, external latches, etc.

### STEADY STATE TRIGGERING

It is possible to trigger a triac with a steady state logic level. This is accomplished by allowing the triac gate to sink or source current during the desired on-time. When utilizing this method it becomes easier to trigger the triac and leave it on for many cycles without having to execute code to retrigger. This approach is advantageous when the triac must be fired for relatively long periods and conduction angle firing is not desired, thus more time is available to accomplish auxiliary tasks. A steady state on or off signal and external circuitry can accomplish triac firing and free the processor for other

tasks. If it is desired to use a pulse transformer, an external oscillator must be gated to the triac to provide the trigger signal. A pulse train of 10 to 15kHz is adequate to fire the triac each half cycle. This calls for external components and is relatively costly. If isolation associated problems can be tolerated or overcome (dual power supply transformers, direct AC coupling, etc.), a simple buffer may be utilized in triggering the triac. This method is illustrated in Figure 8. The National Semiconductor DS8863 display driver is capable of steady state firing of the triac. National offers many buffers capable of driving several hundred milliamps, which are suitable for driving triacs. On the market today there are many suppliers of sensitive gate triacs which may be triggered directly from a COPS device or in conjunction with a smaller external buffer.

The DS8863 display driver is capable of sinking up to 500mA, which is adequate to drive a standard triac. In the off state the driver will not sink current. When a logic "1" is applied to the input the device will turn on. Keeping the device off (output "1") will prevent the triac from turning on because the buffer does not have the capability of sourcing current. A series resistor limits the current from the triac gate and the diode isolates the negative spikes from the gate. Since the drive circuit will only sink current in this configuration, the triac will be operating in the I- and III- modes.

## Triac Light Intensity Control Code

The following code is not intended to be a final functional program. In order to utilize this program, modifications must be made to specialize the routines. This is intended to illustrate the method and is void of control code to command a response such as intensify or de-intensify. The control is up to the user and full understanding of the program must be attained before modifications can be implemented.

This program is a general purpose light intensifying routine which may be modified to suit light dimmer applications. The delay routines require a  $4.469\mu\text{s}$  cycle time which can be attained with a 3.578 MHz crystal (CKI/16 option). This program divides the half cycle of a 60 Hz power line into 16 levels. Intensity is varied by increasing or decreasing the conduction angle by firing the triac at various levels. The program will increase the conduction angle to a maximum specified intensity in a fixed amount of time. The time required to intensify to the maximum level is dependent on the number of fire-times per level that is specified (FINO). This code illustrates a half cycle approach and relies on the parameters specified by the programmer in the control selection.

Zero crossings of the 60 Hz line are detected and software debounced to initiate each half cycle; thus the triac is serviced on every half cycle of the power line. A level/sublevel approach is utilized to vary the conduction angle and provide a prolonged intensifying period. The maximum intensity is specified by the "LEVEL" RAM location and the time required to get to that level is specified by the "FINO" RAM location.

Once a level has been specified, the remaining time in the half cycle is then divided into sublevels. The sublevels are increased in steps to the maximum level. The "FINO" RAM location contains the number of times that the triac will be fired per sublevel, thus creating the intensity time base. There are 15 valid sublevels and up to 15 fire-times per sublevel. Both these parameters may be increased to provide better resolution and longer intensify periods. To make the triac de-intensify (dim) the sublevels need only to be decremented rather than incremented. If this is done, the conduction angle will start out at the maximum level and dim by means of stepping down the sublevels. When modifying this routine to incorporate more resolution or increased versatility, care must be taken to account for transfer of control instructions to and from the delay routines.

The following is a schematic diagram of the COPS interface to 120V<sub>AC</sub> lamps. The program will intensify or de-intensify the lamps under program control.

### TRIAC LIGHT INTENSIFY ROUTINE

This program intensifies a light source by varying the conduction angle applied to the load. The maximum level of intensity is stored in "LEVEL," and the time to get to that level is specified by "FINO." Both these parameters may be altered to suit specific applications. To cause the program to de-intensify the light source, the sublevels must be decremented rather than incremented.

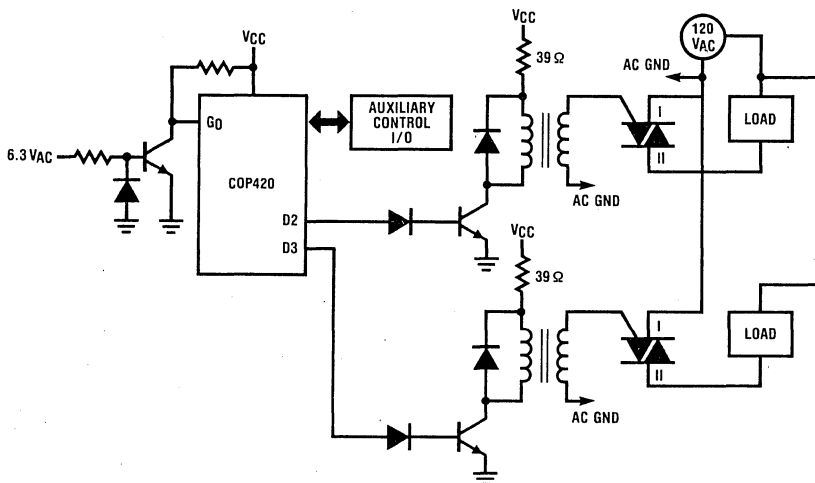


Figure 9. Triac Interface for COPS Program

TRIAI LIGHT INTENSIFY ROUTINE

THIS PROGRAM INTENSIFIES A LIGHT SOURCE BY VARYING THE CONDUCTION ANGLE APPLIED TO THE LOAD. THE MAX LEVEL OF INTENSITY IS STORED IN 'LEVEL' AND THE TIME TO GET TO THAT LEVEL IS SPECIFIED BY 'FIND'. BOTH THESE PARAMETERS MAY BE ALTERED TO SUIT SPECIFIC APPLICATIONS. TO CAUSE THE PROGRAM TO DE-INTENSIFY THE LIGHT SOURCE, THE SUBLEVELS MUST BE DECREMENTED RATHER THAN INCREMENTED.

TEMP1 = 1,0 ; TEMPORARY DELAY COUNTER  
 FIND = 0,9 ; NUMBER OF FIRE TIMES  
 LEVEL = 0,0 ; MAX LEVEL  
 SUBLEV = 1,10 ; SUBLEVEL COUNT  
 TEMP = 1,11 ; TEMPORARY DELAY COUNTER

HERE THE OPERATING PARAMETERS ARE DEFINED AND LEVEL INITIATION IS SPECIFIED

.FORM  
 .PAGE 0  
 CLRAM: CLRA ; REQUIRED  
 CLR: LBI 3,15 ; ROUTINE TO CLEAR ALL RAM  
 XDS CLR  
 JP CLR  
 XABR  
 AISC 15  
 JP BEGG  
 XABR  
 JP CLR

THIS SECTION INITIATES CONTROL ON POWER UP OR RESET AND SYNCHRONIZES THE COPS DEVICE TO THE 60 HZ AC LINE

BEGG: OGI 15 ; OUTPUT 15 TO G PORTS TO PULL  
 ; UP ZERO CROSSER INPUT  
 LBI LEVEL ; SPECIFY MAX LEVEL  
 STII 7  
 JSR OUT ; COPY TO TEMP1  
 BEG: SKGBZ 0 ; SYNC UP TO 60 HZ  
 JP HI ; READY NOW  
 JP BEG ; WAIT TILL G IS 1

THIS SECTION PROVIDES THE DEBOUNCE FOR THE ZERO VOLTAGE DETECTION INPUT AND COMPENSATES FOR THE OFFSET OF THE DETECTION CIRCUIT

HI: SKGBZ 0 ; TEST GO FOR ZERO CROSS  
 JP HI ; HIGH LEVEL  
 ; GETS HERE ON FIRST TRANSITION  
 CLRA ; START OF DEBOUNCE DELAY  
 AISC 1  
 JP -1

DID A LITTLE DELAY, IS IT STILL 0

SKGBZ 0 ; TEST FOR 0  
 JP HI ; FALSE ALARM  
 ; MUST HAVE HAD SOME NOISE GO BACK AND WAIT FOR TRUE ZC  
 DOIT: JMP INT ; VALID TRANSITION, SERVICE  
 ; TRIAC

LO: SKGBZ 0 ; DEBOUNCE IN 0 TO 1  
 JP DDD ; MAY HAVE SOMETHING THERE  
 JP LO ; NO WAIT HERE FOR A BIT  
 DDD: CLRA ; GOING TO WAIT AND SEE  
 AISC 1  
 JP -1

SKGBZ 0 ; WELL, DO WE HAVE A CLEAN  
 ; TRANSITION  
 JP DELL ; YES, GO TO MAIN ROUTINE  
 JP LO ; FALSE ALARM, TRY AGAIN

DELL: CLRA ; DO A DELAY TO COMPENSATE  
 DEL: NOP ; FOR NON SYMMETRIC ZC  
 NOP  
 NOP

AISC 1  
 JP DEL ; KEEP DELAY GOING  
 JP DOIT ; GO TO MAIN ROUTINE

.FORM  
 .PAGE 1

THIS IS THE MAIN ROUTINE FOR THE INTENSIFY/DE-INTENSIFY OPERATIONS. TRANSFER OF CONTROL TO THIS SECTION OCCURS AFTER ZERO VOLTAGE CROSSING EACH HALF CYCLE. THIS MAKES USE OF TEMP REGISTERS THUS PARAMETERS

NEED NOT BE REDEFINED FOR EACH OPERATION.

INT: CLRA ; DELAY INTO WAVEFORM  
 ADT ; USE TEMP REG  
 LBI TEMP  
 X ; DO DELAY  
 JSRP PORT ; POINT TO LEVEL TO INITIATE  
 POINT: LDD LEVEL ; DELAY  
 ; DELAY TO MAX LEVEL  
 ; USE TEMP DIGIT TO DELAY  
 TEMP: XAD TEMP  
 LBI TEMP  
 LD ; ARE WE AT THE LEVEL ?  
 AISC 15 ; MADE IT TO THE LEVEL  
 JP ATLEV ; NO  
 X ; DO SERIES OF .5MS TO GET  
 JSRP DE5 ; THERE  
 ; KEEP DOING IT  
 ATLEV: JP TAMP ; AT MAX FIRE LEVEL  
 LDD SUBLEV ; INIT FOR SUBLEVEL DELAY  
 XAD TEMP  
 JK: LBI TEMP  
 LD ; AT SUB LEVEL ?  
 AISC 1 ; NO DO DELAY  
 JP TRE ; YES  
 JP SBLEV  
 TRE: X ; VARIABLE DELAY  
 JSRP SPDL  
 JP JK  
 SBLEV: LBI FIND  
 JSRP DEC ; DEC FIRE NUMBER  
 AISC 1 ; TEST IF FIND AT 15  
 JMP FIRE ; NO KEEP FIRING AT THAT LEVEL  
 LBI SUBLEV ; YES INC SUBLEVEL  
 CLRA  
 AISC 14 ; IS MAX SUBLEV REACHED  
 SKE  
 JP THERE ; NO INC SUBLEV  
 JP MAXLEV ; YES FIRE IT  
 THERE: JSRP INC ; GO TO NEXT SUBLEVEL  
 LBI FIND  
 STII 14 ; SET FIRE TIME  
 JP MAXLEV ; GO FIRE

.FORM  
 .PAGE 2  
 ; SUBROUTINE PAGE

INC: CLRA  
 AISC 1  
 JP ADEX ; GO ADD ONE TO DIGIT  
 DEC: CLRA ; 0 TO A  
 COMP ; CREATE A 15  
 ADD ; ADD A TO RAM  
 X ; PUT BACK (D-1 IN A NOW)  
 ADEX: X  
 RET  
 DE5: LBI 0,10 ; DELAY ROUTINE  
 CLRA ; WILL BE REPLACED LATER  
 AISC 3  
 JP -1  
 LD  
 XIS  
 JP -5  
 FIRE: RET ; DONE DELAY  
 LBI 0,15 ; PULSE D OUTPUT  
 OBD  
 NOP  
 NOP  
 NOP  
 LBI 0,0  
 OBD  
 SKGBZ 0 ; TEST WHICH DEBOUNCE IS  
 ; NEEDED  
 JMP HI ; DEBOUNCE ONE TO ZERO  
 JMP LO ; DEBOUNCE ZERO TO ONE  
 SPDL: LBI TEMP1 ; TEMP1 IS A TEMP REG  
 PORT: LD ; VALUE IN TEMP1 DICTATES  
 AISC 1 ; THE AMOUNT OF DELAY  
 JP FOY  
 OUT: LBI LEVEL ; ALSO USED TO COPY LEVEL  
 LD 1 ; RESTORE LEVEL  
 X  
 RET  
 X  
 JP PORT  
 .END



## Table of Contents

I. Introduction	9-159
II. Philosophy	9-159
III. Built-in Test Features	9-159
A. Sync between DUT and Tester	9-160
B. Internal Logic Test	9-160
C. RAM Test	9-161
D. ROM Dump	9-161

## Testing of COPS™ Chips

This note will provide some insight into the test mode, the mechanics of testing, and the philosophy of how to implement a test of the COP-400 microcontrollers. Other than the obvious, (verifying that the part meets the specifications), the reason for the test must be considered. Somewhat different criteria may hold, depending on the objective. The manufacturer wafer sort or final test can differ from an incoming inspection at the user's plant, or a field reject test. The first two tests have limited interest as this is not a justification of the testing done on the part during manufacture. Rather, this is a guide for those doing user functional testing.

### I. Introduction

Since the introduction of the very first semiconductor devices, testing has been a major problem and expense in their production and use. As the complexity has risen, testing has become a more significant factor. With today's single chip microcontrollers like the COPS™ devices this is particularly true as one has a complete computer system in a chip. In order to reduce the testing burden, the facilities to ease the testing have been built into the COPS™ devices. With the test ability built into the device for production test, the user need only follow set procedures to verify the chip at incoming inspection or field test.

### II. Philosophy

The basic test philosophy requires that four major areas be exercised. These areas are:

- 1) Synchronize the device and tester.
- 2) Test the internal logic and I/O.
- 3) Test the RAM
- 4) Verify the ROM program.

If the devices perform all of these four properly, the device is good. This is a reasonable assumption with a standard device that has a debugged test routine and is ROM programmed. A custom circuit just going into production might not have the accumulated test background. By attacking the problem on a "sum of the parts" approach, one need not do any exhaustive functional test on routine production parts. This will be a major gain where lengthy time consuming or time dependent routines are involved. If one attempts to do a functional test of the chip, a sequence that is unique to the application is needed. Thus, a test program must be written and debugged for each ROM pattern. Further, a test box/board must be designed, built, debugged, documented, and maintained for each one. If testing has been considered from the beginning, the chip will have built-in capabilities to exercise the various parts of it. The different functional parts and instructions are tested to verify proper operation at the voltage and frequency limits.

### III. Built-in Test Features

The first step in testing the COP400 devices is to understand the built-in test control features. This will involve the SI/O and the L lines. The SO pin has been designed to be the control node for testing. The pin will normally be in an active low state and when forced high externally, places the chip in the test mode. It should be noted that this output can sink considerable current and one should not force the pin to the  $V_{CC}$  rail. By limiting the voltage to the 2.0/3.0V range one can not damage the device where the application of a higher voltage could. When forced into the test mode the SI pin controls the sub mode of the chip. With SI high the data placed on the L port is used as an instruction. When SI is low (and the L output is enabled) the contents of the ROM will be dumped out through the L port. Certain other internal functions have been implemented to allow these modes but these are not part of the basic operation. Included in this category is the activation of the skip signal to prevent the program counter from jumping out of sequence by executing a program control instruction.





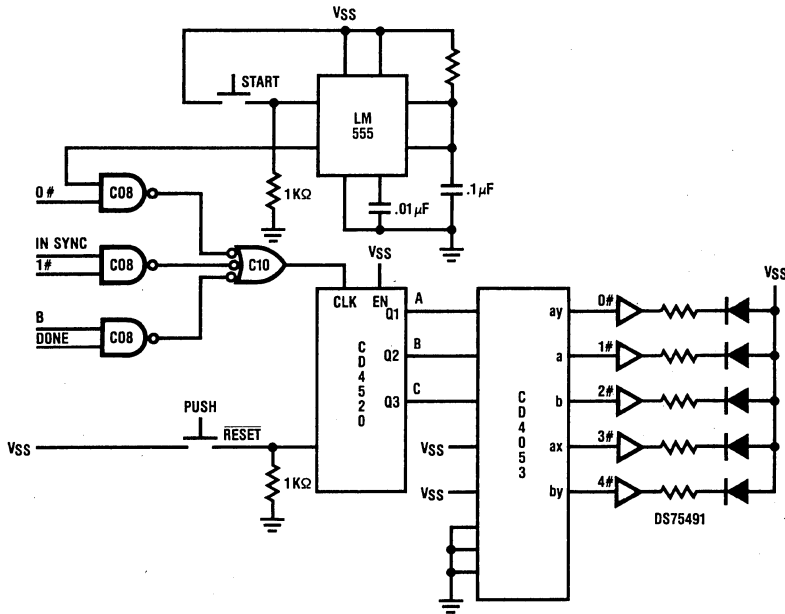


Figure 2. Tester Mode Sequencer

### C. RAM Test

The verification of RAM is a part of the internal logic test, but is treated separately here. One must check both the RAM and its address register to find all faults. An example of this testing would be to load RAM with a string of STII commands. By then going back and reading this data to the outside (through an OMG instruction in a loop) the tester could verify both RAM and address were functional. One could then load RAM with all 6's and 9's (or 5's and 10's) sequentially to insure that all bits were functional and adjacent bits not shorted. Other similar tests could be run at the discretion of the user to do further testing. All of these tests would utilize the output of data via the G ports to validate the data. See the comparator circuit Figure 3.

### D. ROM Dump

Successful operation of the internal logic tests and RAM will lead to the final test phase, ROM comparison. In order to check the ROM contents, the ROM dump

mode must be entered. One should force a JMP to an address near the end of the ROM space (3FF for a 420 chip, 1FF for a 410). A desirable point might be 3FA. The program counter will step ahead on each instruction cycle unless a program control is executed. The next step is to load the Q register with a non-conflicting value so that the enabling of the L outputs will not destroy the second byte of the LEI instruction as control is passed into the ROM dump mode. After going to this address, one should execute an enable of the L lines to the output port (LEI 4). Having done this the external buffers should be disabled and the SI pin taken low. This will allow data out and remove potential level conflicts. By letting the PC step ahead to address zero one can then begin the byte by byte comparison of data. In this mode the controller is not executing the code because the skip line is enabled throughout the sequence. By halting a counter on a failure, one could determine the questionable address.



Table 1. Typical Test Sequence

INSTRUCTION	RESULT	COMMENTS	INSTRUCTION	RESULT	COMMENTS
NOP	NO CHANGE	CHECK NOP & ALLOW TRANSIENT CYCLE FOR MODE	OMG	G(9 > 1)	
OGI 9	G(0 > 9)	NOT ON 410L/411L	LD 3		1 > A; Bd > 2,0
OGI 6	G(9 > 6)	REVERSE ALL G STATES	OMG	G(1 > 2)	
STII 8		SET UP 0,0 FOR FUTURE	ADD		ADD WITHOUT CARRY
LBI 3,13		B TO NEW POSITION (3,13)	X		STORE 3 IN 2,0
OBd	D(0 > 13)	CHECK D	SC		7 > A
CLRA		MAKE SURE A=0	LDD 0,0		CHECK CASC
XABR		3 > A; 0 > Br	CASC		
CAB		MOVE 3 to Bd	SKC		STORE 12
OBd	D(13 > 3)	CHECK XABR CAB & D CHANGE	X		
CLRA		!	OMG	G(2 > 12)	
.AISC 2		FORCE A > 2	CLRA		:
CAB		2 > Bd	AISC 3		:
OBd	D(3 > 2)	VERIFY 2 FROM A > Bd	X		:
STII 7		7 > 0,2 & Bd > 3	SC		: CHECK
OBd	D(2 > 3)	STII INCREMENTS Bd	SKC		: SKC/SC
CAB		SEE THAT A STILL THE SAME	X		:
OMG	G(6 > 7)	OMG & RAM CHECK	OMG	G(12 > 3)	
CLRA			RC		
CAB		B(0,0)	SKC		: CHECK
OMG	G(7 > 8)	TIE IN RAM, A & G OPERATION	X		: RC
SMB 0		SMB INST. CHECK	OMG	G(3 > 12)	
OMG	G(8 > 9)	:	LBI 0,0		: CHECK
SMB 1		:	LBI 1,15		: SEQUENTIAL LBI'S
OMG	G(9 > 11)	:	LBI 2,7		: ALSO SKIPPED (LBI 2,7 NOT IN 410)
RMB 0		:	OMG	G(2 > 7)	
RMB 3		:	CQMA		: LOAD CONSTANTS FROM Q
X		: 0 > 0,0; 2 > A	OMG	G(7 > 9)	: CHECK
CAB		A = 2 > B	X		:
OMG	G(11 > 7)	OUTPUT M(0,2)	OMG	G(9 > 10)	
LD 1		M(0,2) > A; B > 1,2	LEI 1		
XAD 0,0		A(7) <-> M(0,0) 2	XAS		STORE A - > S (9)
AISC 15		AISC CHECK; A=1	CLRA		
LDD 0,0		CHECK SKIP OF 2 BYTE INST.	AISC 7		
X		STORE 1	SKGBZ 0		
OMG	G(7 > 1)	VERIFY	X		: CHECK
LD 0		COPY 1,2 BACK TO A	OMG		: CHECK
ADT		ADD TEN	SKGBZ 1		: G BIT
XDS		LEAVE 11 IN 1,2; GO 1,1 WITH 1	X		
XDS		LEAVE 1 IN 1,1; GO 1,0 W ?	OMG	G(10 > 7)	
OBd	D(2 > 0)	CHECK Bd MOVEMENT	SKGBZ 2		
STII 5		5 > 1,0; Bd to 1,1	X		
CBA		CHECK B > A	OMG	G(7 > 10)	: TESTS
AISC 3		AISC CHECK 4 > A	SKGBZ 3		
			X		
			OMG	G(10 > 7)	
INSTRUCTION	RESULT	COMMENTS	INSTRUCTION	RESULT	COMMENTS
XDS		1 > A; 4 > 1,1	SKGZ		: CHECK
OMG	G(1 > 5)	FROM 1,0	X		:
XDS		5 > A; 1 > 1,0; Bd < 15 SKIP	OMG	G(7 > 10)	
LDD 0,0	D(0 > 15)	SKIPPED !	OGI 0	G(10 > 0)	: G TEST
OBd			SKGZ		:
AISC 4		9 > A	X		
X		9 > 15	OMG	G(0 > 10)	
OMG	G(5 > 9)		SKMBZ 0		: CHECK MEMORY BIT TESTS
CLRA		ONES TO A	X		: NO CHANGE
COMP		FLIP MEMORY	OMG		
XOR		6 > 1,15; 9 > A; Bd > 1,0	SKMBZ 1		
XIS		SKIP	X		
LDD 0,0			OMG	G(10 > 7)	: NO SKIP
SKE			SKMBZ 2		
LB 1,2		SKIP 2 WORD LBI (NOT IN 410)	X		: WON'T SKIP
OBd	D(15 > 0)	VERIFY WORD	OMG	G(7 > 10)	: SEE THAT L LATCHES RESET
SKE		11 NOT=9	ININ		: ASSUME G - > I
LBI 1,0		BACK TO 1,0	SKE		
SMB 2			X 1		: Br > 1
SKE			OMG		: SHOULD BE EQUAL
RMB 2			INIL		
SKE		: CHECK BIT	X		
SMB 3		: MANIPULATIONS	SKMBZ 3		
SKE			OBd	D(15 > 0)	: INIL TEST
LDD 0,0			OGI 1		
X 3		Bd > 2,0	LBI 3,11		
XAD 1,1		9 > 1,1; 4 > A	OGI 0		
XIS 1		4 > 2,0; Bd > 3,1	INIL		
ING		INPUT G PORT	X		
X		STORE	SKMBZ 0	D(0 > 11)	
CLRA			OBd		
ASC		CHECK ADD WITH CARRY	NOP		
SC		CHECK SET CARRY	XAS		: XAS TEST
SKC		CHECK SKIP ON CARRY	X		
LDD 0,0			OMG	G(10 > 9)	
X		STORE A			
OMG	G=9	NO CHANGE			
CLRA					
ASC					
X					
OMG	G(9 > 10)	CARRY ADDS ONE TO MEMORY			
CAMQ		STORE A & M IN Q; 10,9			
XDS		9 > 3,1; 10 > A; Bd > 3,0			
X		STORE 9 IN 3,0			
OMG	G(10 > 9)				
LD 2		9 > A; Bd > 1,0			





# SIO Input/Output Register Description

National Semiconductor  
COP Brief 1  
May 1980



## Contents

- Logical Operation
- Software Debug
- Serial Out During Breakpoint
- Serial Out During Trace
- Binary Counter During Breakpoint
- General
- Using SIO as temporary storage

## COP400 Serial SIO Register

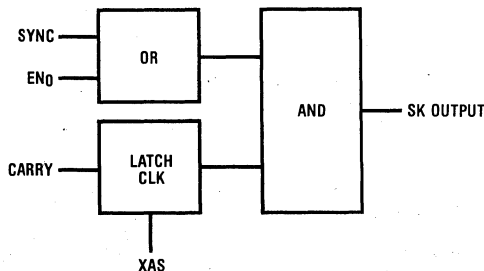
The general operation of the SIO port is treated in the COP400 data sheet. A more detailed look at the internal circuitry, as well as software debug, will be presented in this brief.

### Logical Operation

It is important to examine the logical diagram of the SIO and SK circuitry to fully understand the operation of this I/O port. The output at SK is a function of SYNC, EN<sub>0</sub>, CARRY, and the XAS instruction.

If CARRY had been set and propagated to the SKL latch by the execution of an XAS instruction, SYNC is enabled to SK and can only be overridden by EN<sub>0</sub>. Trouble could arise if the user changes the state of EN<sub>0</sub> without paying close attention to the state of the latch in the SK circuit.

If the latch was set to a logical high and the SIO register enabled as a binary counter, SK is driven high. From this state, if the SIO register is enabled as a serial shift register, SK will output the SYNC pulse immediately, without any intervening XAS instruction.



Logical Diagram of SK Circuit

## Software Debug of Serial Register Functions

In order to understand the method of software debug when dealing with the SIO register, one must first become familiar with the method in which the COPS Product Development System (PDS) BREAKPOINT and TRACE operations are carried out. Once these operations are explained, the difficulties which could arise when interrogating the status of the SIO register should become apparent.

### Serial Out During BREAKPOINT

When the PDS BREAKPOINTS, the COPS user program execution is stopped and execution of a monitor-type program, within the COP device is started. At no time does the COP part "idle". The monitor program loads the development system with the information contained in the COP registers.

Note also that single-step is simply a BREAKPOINT on every instruction.

If the COP chip is BREAKPOINTed while a serial function is in progress, the contents of the SIO register will be destroyed. By the time the monitor program dumps the SIO register to the PDS, the contents of the SIO register will have been written over by clocking in SI. To inspect the SIO register using BREAKPOINT an XAS must be executed prior to BREAKPOINT, therefore the SIO register will be saved in the accumulator.

An even more severe consequence is that the monitor program executes an XAS instruction to get the contents of the SIO register to the PDS. Therefore the SK Latch is dependent on the state of the CARRY prior to the BREAKPOINT. In order to guarantee the integrity of the SIO register one must carefully choose the position of the BREAKPOINT address.

As can be seen, it is impossible to single-step or BREAKPOINT through a serial operation in the SIO register.

### Serial Out During TRACE

In the TRACE mode, the user's program execution is never stopped. This mode is a real-time description of the program counter and the external event lines, therefore the four external event lines can be used as logic analyzers to monitor the state of any input or output on the COPS device. The external event lines must be tied to the I/O which is to be monitored. The state of these I/O (External Event lines) is displayed along with the TRACE information. The safest way to monitor the real-time state of SO is to use the TRACE function in conjunction with the External Event lines.

### Binary Counter During BREAKPOINT

Since the COPS chip is executing a Monitor Program during BREAKPOINT the SIO register is still active. In the Binary Counter mode SIO register will decrement on every negative transition of the SI line providing the pulse

stays low for at least two instruction cycles. However, if the pulse on SI occurs when the monitor is interrogating the SIO register, an erroneous situation may occur.

### General

During a BREAKPOINT operation data is transmitted to the PDS over the SKIP output on the COP402.

Notice that the D register is not contained in the Auto-Print options. The reason for this is that the contents of D cannot be read via COP software. These may be monitored by the External Event lines in the trace mode.

### Temporary Storage

It is sometimes desirable to temporarily store the value of the accumulator. This can be done by designating a RAM digit and doing an exchange operation. If the user can assure that the SIO register is in the binary counter mode and that SI is at a constant state, the SIO register may be used as a temporary storage location. This is advantageous because the storage and retrieval is accomplished by the single byte XAS instruction and does not require the use of a RAM digit. The use of the SIO register as a binary counter is not available on the COP420C (CMOS version of the COP420), for this reason the SIO register may not be used as temporary storage.



# Easy Logarithms for COP400

National Semiconductor  
COP Brief 2  
May 1980

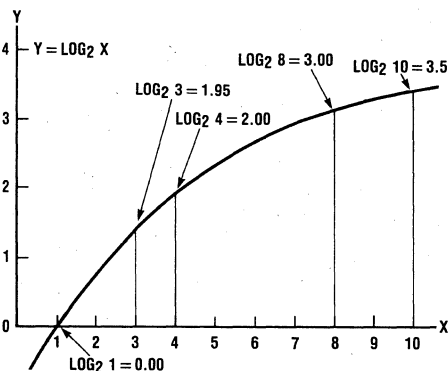
Logarithms have long been a convenient tool for the simplification of multiplication, division, and root extraction. Many assembly language programmers avoid the use of logarithms because of supposed complexity in their application to binary computers. Logarithms conjure up visions of time consuming iterations during the solution of a long series. The problem is far simpler than imagined and its solution yields, for the applications programmer, the classical benefits of logarithms:

- 1) Multiplication can be performed by a single addition.
- 2) Division can be performed by a single subtraction.
- 3) Raising a number to a power involves a single multiply.
- 4) Extracting a root involves a single divide.

When applied to binary computer operation logarithms yield two further important advantages. First, a broad range of values can be handled without resorting to floating point techniques (other than implied by the characteristic). Second, it is possible to establish the significance of an answer during the body of a calculation, again, without resorting to floating point techniques.

Implementation of base<sub>10</sub> logarithms in a binary system is cumbersome and unnecessary since logarithmic functions can be implemented in a number system of any base. The techniques presented here deal only with logarithms to the base<sub>2</sub>.

A logarithm consists of two parts: an integer characteristic and a fractional mantissa.



	CHARACTERISTIC	MANTISSA
LOG <sub>2</sub> 3 =	1	0.95
LOG <sub>2</sub> 4 =	2	0.00
LOG <sub>2</sub> 8 =	3	0.00
LOG <sub>2</sub> 10 =	3	0.52

Figure 1. The logarithmic function and some example values

In figure 1 some points on the logarithmic curve are identified and evaluated to the base<sub>2</sub>. Notice that the characteristic in each case represents the highest even power of 2 contained in the value of X. This is readily seen when binary notation is used.

X <sub>10</sub>	X <sub>2</sub>					Log <sub>2</sub> X Characteristic	Log <sub>2</sub> X Where X = Even Power of 2
	2 <sup>4</sup>	2 <sup>3</sup>	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>		
3	0	0	0	1	1	1	
4	0	0	1	0	0	2	010.0000
8	0	1	0	0	0	3	011.0000
10	0	1	0	1	0	3	

Figure 2. Identification of the Characteristic

In Figure 2 each point evaluated in Figure 1 has been repeated using binary notation. An arrow subscript indicates the highest even power of 2 appearing in each value of X. Notice that in X = 3 the highest even power of 2 is 2<sup>1</sup>. Thus the characteristic of the log<sub>2</sub> 3 is 1. Where X = 10 the characteristic of the log<sub>2</sub> 10 is 3.

To find the log<sub>2</sub> X is very easy where X is an even power of 2. We simply shift the value of X left until a carry bit emerges from the high order position of the register. This procedure is illustrated in Figure 3. This characteristic is found by counting the number of shifts required and subtracting the result from the number of bits in the register. In practice it is easier to begin with the number of bits and count down once prior to each shift.

Counter For Characteristic	Value of X in Binary	
1 0 0 0	0 0 0 0 1 0 0 0	Initial
0 1 1 1	0 0 1 0 0 0 0	First Shift
0 1 1 0	0 0 1 0 0 0 0	Second Shift
0 1 0 1	0 1 0 0 0 0 0	Third Shift
0 1 0 0	1 0 0 0 0 0 0	Fourth Shift
0 0 1 1	0 0 0 0 0 0 0	Fifth Shift
<b>Characteristic</b>	<b>Mantissa</b>	<b>Final</b>
0 1 1 . 0 0 0 0	0 0 0 0	Log <sub>2</sub> X = 3.00

Figure 3. Conversion to Base<sub>2</sub> Logarithm by Base Shift

Examination of the final value obtained in Figure 3 reveals no bits in the mantissa. The value 3 in the characteristic; however, indicates that a bit did exist in the 2<sup>3</sup> position of the original number and would have to be restored in order to reconstruct the original value (antilog).

The log of any even power of 2 can be found in this way:

Decimal	Binary	Log <sub>2</sub>
128	10000000	01111.00000000
64	01000000	0110.00000000
32	00100000	0101.00000000
4	00000100	0010.00000000
2	00000010	0001.00000000
1	00000001	0000.00000000

A simple flow chart, and program, can be devised for generating the values found in the table and, as will be apparent, a straight line approximation for values that are not even powers of 2. The method, as already illustrated in Figure 3, involves only shifting a binary number left until the most significant bit moves into the carry position. The characteristic is formed by counting. Since a carry on each successive shift will yield a decreasing power of 2, we must start the characteristic count with the number of bits in the binary value (x) and count down one each shift.

Figure 4. Base<sub>2</sub> Logarithms of Even Powers of 2

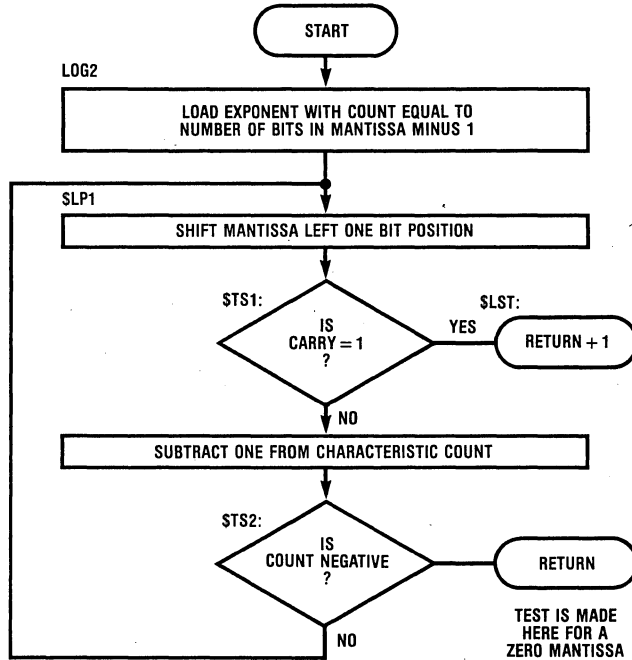


Figure 5. Log Flowchart

COP CROSS ASSEMBLER .PAGE: 1  
LOGS

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72

01A4

; TITLE LOGS ; BINARY LOGARITHMS

. CHIP 420

; ----- CONVERT TO LOGARITHM -----;

RAM ASSIGNMENT

DIGIT:	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
REG 0									CH	HM	LM					TEMP
REG 1				CH	HM	LM										TEMP
REG 2						TEMP								CH	HM	LM
REG 3										TEMP				CH	HM	LM

. LOCAL

; CH, HM, LM REPRESENT ANY THREE SEQUENTIAL MEMORY DIGITS. THEY  
; MAY BE DEFINED IN ANY REGISTER. THE SYMBOLIC NOTATION CH, HM,  
; AND LM ARE USED FOR ADDRESSING TO ALLOW USER FLEXIBILITY.  
; UPON ENTRY TO THE ROUTINE HM AND LM CONTAIN THE HI AND LO  
; OF SOME VALUE X. THE MEMORY POINTER MUST CONTAIN THE ADDRESS  
; OF THE CHARACTERISTIC (CH). THE CONTENTS OF THIS LOCATION ARE  
; IGNORED AND ARE LOST DURING EXECUTION.

; UPON EXIT CH, HM, LM CONTAIN A STRAIGHT LINE APPROXIMATION OF  
; THE LOG BASE 2 OF X. CH = CHARACTERISTIC HM = HI ORDER MANTISSA  
; LM = LO ORDER MANTISSA. AN 8 BIT MEMORY AREA (TEMP) IS USED IN  
; THE REGISTER OPPOSITE DURING THE CORRECTION OF A STRAIGHT  
; LINE APPROXIMATION OF A LOG OR AN ANTILOG.

; A TEST IS MADE FOR X=0. IF THE VALUE OF X  
; IS NOT ZERO AN INSTRUCTION IS SKIPPED UPON RETURN  
; TO THE CALLING ROUTINE.

— EXAMPLE —

```

SUBROUTINE CALL                JSR LOG2
RETURN HERE IF X=0 ---->      JP ZERO
RETURN HERE IF X>0 ---->      CONTINUE
    
```

```

LOG2:  CLRA                    ; SET CHARACTERISTIC.
        AISC                    ; TO REG LENGTH -1.
        X                       ; STORE IN MEMORY.
    
```

COP CROSS ASSEMBLER .PAGE: 2  
LOGS

```

53 003 A4    $LP1:  JSRP        SDB2        ; SET ADDRESS POINTER
54                                ; BACK 2 DIGITS.
55 004 A9                                ; RESET CARRY AND SHIFT
56                                ; REG LEFT ONE BIT.
57 005 20    $TS1:  SKC          $NO        ; IS CARRY = 1 YET?
58 006 C8                                ; NO — KEEP GOING.
59 007 49    $LST:  RETSK       ; YES — FINISHED!!
60 008 05    $NO:   LD          ; NO — LOAD COUNT IN ACC.
61 009 5F                                ; SUBTRACT ONE.
62 00A 48    $TS2:  RET          ; MANTISSA IS A 0! RETURN
63 00B 06                                ; STORE CHARACTERISTIC.
64 00C C3                                ; DO IT AGAIN!
        X
        JP          $LP1
    
```

; 2 ROUTINES ARE CALLED FROM THE SUBROUTINE PAGE BY THIS  
; PROGRAM: SDB2, SHLR.

Figure 6.

The program shown develops the  $\log_2$  of any even power of 2 by shifting and testing as previously described. Examine what happens to a value of X that is not an even power of 2. In Figure 7, the number 25 is converted to a base 2 log.

$$25_{10} = 00011001_2$$

Shift left until carry = 1

Characteristic	Carry	Mantissa	Log <sub>2</sub>
0100	1	10010000	0100.10010000

Figure 7. Straight Line Approximation of a Base<sub>2</sub> Log

The resulting number when viewed as an integer characteristic and fractional mantissa is 4.5625<sub>10</sub>. The fraction 0.5625 is a straight line approximation of the logarithmic curve between the correct values for the base<sub>2</sub> logs of 2<sup>4</sup> and 2<sup>5</sup>. The accuracy of this approximation is sufficient for many applications. The error can be corrected, as will be seen later in this discussion, but for now let's look at the problem of exponents or the conversion to an antilog.

To reconstruct the original value of X, find the antilog, requires only restoration of the most significant bit and then its alignment with the power of 2 position indicated by the characteristic. In the example, approximation ( $\log_2 25 = 0100.1001$ ) restoration of MSB can be accomplished by shifting the mantissa (only) one position to the right. In the process a one is shifted into the MSB position.

Approximation of Log <sub>2</sub> X		Restoration of MSB	
Char.	Mantissa	Char.	Mantissa
0100	10010000	0100	11001000

The value of the characteristic is 4 so the mantissa must be shifted to the right until MSB is aligned with the 2<sup>4</sup> position.

2 <sup>7</sup>	2 <sup>6</sup>	2 <sup>5</sup>	2 <sup>4</sup>	2 <sup>3</sup>	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>
0	0	0	1	1	0	0	1

The completion of this operation restores the value of X (X = 25) and is the procedure used to find an antilog. Figure 8 is a flow chart for finding an antilog using this procedure. The implementation in source code is shown in Figure 9.

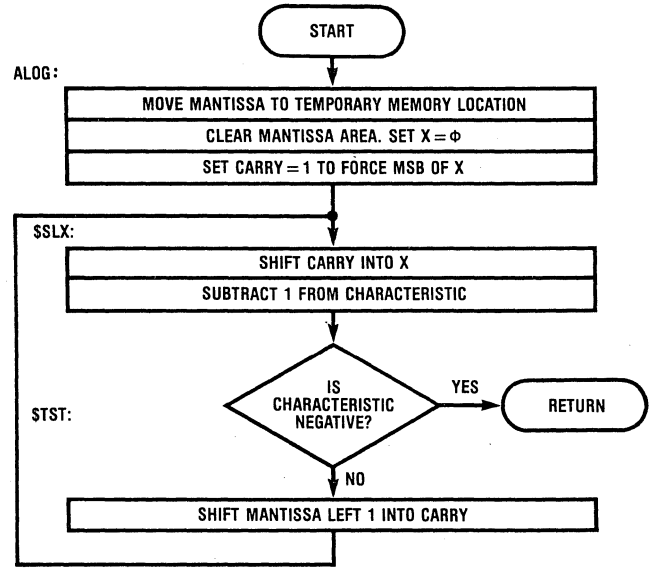


Figure 8. Flow Chart for Conversion to Antilog

```

73          .FORM          ;----- CONVERT TO ANTILOG -----;
74
75
76          ; THE FOLLOWING SUBROUTINE CONVERTS THE STRAIGHT LINE
77          ; THE APPROXIMATION OF A BASE 2 LOGARITHM TO ITS CORRESPONDING
78          ; ANTILOG. UPON EXIT FROM THE ROUTINE THE CONTENTS OF CH
79          ; WILL BE EQUAL TO THE HEXADECIMAL VALUE OF '0F'.
80
81          .LOCAL
82
83
84 00D A4          ALOG:      JSRP          SDB2          ; SET ACC TO 0.
85 00E 00          CLRA          ; CLEAR MANTISSA AREA.
86 00F 36          X          03          ; AND MOVE MANTISSA TO
87 010 34          XIS          03          ; TEMPORARY STORAGE.
88 011 00          CLRA          ; LEAVE POINTER AT LO
89 012 36          X          03          ; ORDER OF MANTISSA.
90 013 37          XDS          03
91 014 22          SC          ; RESTORE MSB OF X.
92 015 D8          JP          $SLX
93 01 A9          $SLM:      JSRP          SHLR          ; SHIFT REMAINDER
94                                     ; LEFT INTO CARRY.
95 017 A3          JSRP          SDR2          ; MOVE BACK 2 DIGITS.
96 018 AA          $SLX:      JSRP          SHLC          ; SHIFT X LEFT 1.
97 019 05          LD          ; LOAD CHARACTERISTIC.
98 01A 5F          $TST:      AISC          -1          ; CHARACTERISTIC - 1.
99 01B 48          $LST:      RET          ; IF NO CARRY — FINIS.
100 01C 36         X          03          ; STORE REMAINDER AND MOVE
101                                     ; DOWN ONE REGISTER.
102 01D A4          JSRP          SDB2          ; MOVE BACK 2 DIGITS.
103 01E D6         JP          $SLM          ; DO IT AGAIN.
104
105
106          ; 4 ROUTINES ARE CALLED FROM THE SUBROUTINE PAGE BY THIS
107          ; PROGRAM: SDB2, SDR2, SHLR, SHLC.
108
109

```

Figure 9.

Using the linear approximation technique just described, some error will result when converting any value of X that is not an even power of 2.

Figure 10 contains a table of correct base 2 logarithms for values of X from 1 through 32 along with the error incurred for each when using linear approximation. Notice that no error results for values of X that are even powers of 2. Also notice that the error incurred for multiples of even powers of 2 of any given value of X is always the same:

Value of X	Error
5	0.12
2 × 5 = 10	0.12
4 × 5 = 20	0.12
3	0.15
2 × 3 = 6	0.15
4 × 3 = 12	0.15
8 × 3 = 24	0.15

X	Hexadecimal Log Base	Linear Approximation of Log Base 2	Error in Hexadecimal	$E_M - 1 + \frac{E_M - E_{M-1}}{2}$
1	0.00	0.00	0.00	
2	1.00	1.00	0.00	
3	1.95	1.80	0.15	
4	2.00	2.00	0.00	
5	2.52	2.40	0.12	
6	2.95	2.80	0.15	
7	2.CE	2.C0	0.0E	
8	3.00	3.00	0.00	
9	3.2B	3.20	0.0B	
10	3.52	3.40	0.12	
11	3.75	3.60	0.15	
12	3.95	3.80	0.15	
13	3.B3	3.A0	0.13	
14	3.CE	3.C0	0.0E	
15	3.E8	3.E0	0.08	
16	4.00	4.00	0.00	
17	4.16	4.10	0.06	0.03
18	4.2B	4.20	0.0B	0.09
19	4.3F	4.30	0.0F	0.0D
20	4.52	4.40	0.12	0.11
21	4.67	4.50	0.17	0.15
22	4.75	4.60	0.15	0.16
23	4.87	4.70	0.17	0.16
24	4.95	4.80	0.15	0.16
25	4.A4	4.90	0.14	0.15
26	4.B3	4.A0	0.13	0.14
27	4.C1	4.B0	0.11	0.12
28	4.CE	4.C0	0.0E	0.10
29	4.DB	4.D0	0.0B	0.0D
30	4.E8	4.E0	0.08	0.0A
31	4.F4	4.F0	0.04	0.06
32	5.00	5.00	0.00	0.02
33		5.1-		

Figure 10. Error Incurred by Linear Approximation of Base 2 Logs

An error that repeats in this way is easily corrected using a look-up table. The greatest absolute error will occur for the least value of X not an even power of 2,  $x=3$ , is about 8%. A 4 point correction table will eliminate this error but will move the greatest uncompensated error to  $X=9$  where it will be about 4%. This process

continues until at 16 correction points the maximum error for the absolute value of the logarithm is less than 1 percent. This can be reduced to 0.3 percent by distributing the error. Interpolated error values are listed in Figure 10 and are repeated in Figure 11 as a binary table.

High Order 4 Mantissa Bits	Binary Correction Value	Hexadecimal Correction Value
0000	0000 0000	0 0
0001	0000 1001	0 9
0010	0000 1101	0 3
0011	0001 0001	1 1
0100	0001 0101	1 5
0101	0001 0110	1 6
0110	0001 0110	1 6
0111	0001 0110	1 6
1000	0001 0101	1 5
1001	0001 0100	1 4
1010	0001 0010	1 2
1011	0001 0000	1 0
1100	0000 1101	0 D
1101	0000 1010	0 A
1110	0000 0110	0 6
1111	0000 0010	0 2

Figure 11. Correction Table for L<sub>2</sub> X Linear Approximations

Notice in Figure 10 that left justification of the mantissa causes its high order four bits to form a binary sequence that always corresponds to the proper correction value. This works to advantage when combined with the COP400 LQID instruction. LQID implements a table look-up function using the contents of a memory location as the address pointer. Thus we can perform the required table look-up without disturbing the mantissa.

Figure 12 is the flow chart for correction of a logarithm found by linear approximation. Figure 13 is its implementation in COP400 assembly language. Notice that there are two entry points into the program. One is for correction of logs (LADJ); the other is for correction of a value prior to its conversion to an antilog (AADJ).

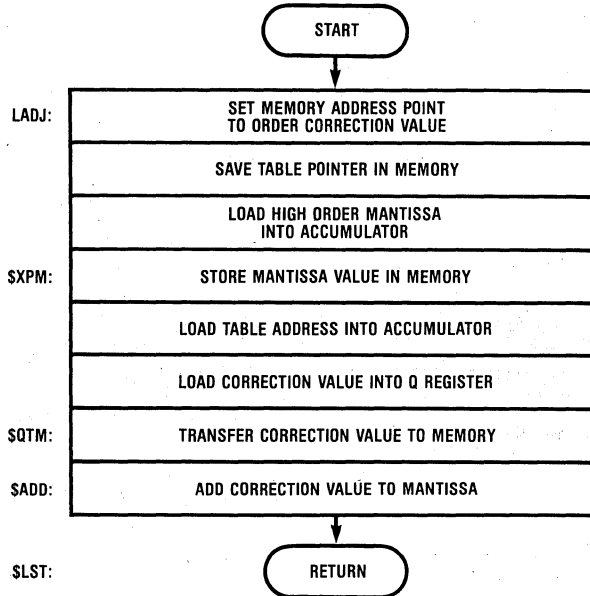


Figure 12. Flow Chart for Correction of a Value Found by Straight Line Approximation

COP CROSS ASSEMBLER PAGE: 4  
LOGS

```

110          . FORM          ; ----- ADJUST VALUE OF LOGARITHM ----- ;
111
112          . LOCAL
113
114
115          ; THE FOLLOWING TABLE IS USED DURING THE CORRECTION OF VALUES
116          ; FOUND BY STRAIGHT LINE APPROXIMATION. IT IS PLACED HERE IN
117          ; ORDER TO ALIGN ITS BEGINNING ELEMENT WITH A ZERO ADDRESS AS
118          ; REQUIRED BY THE LQID INSTRUCTION.
119
120 01F 44          NOP          ; REGISTER WITH ZERO ADDRESS.
121 020 03  TPLS:  . WORD      03,09,0D,011
122 021 09
123 022 0D
124 023 11
125 024 15  . WORD      015,016,016,016
126 025 16
127 026 16
128 027 16
129 028 15  . WORD      015,014,012,010
130 029 14
131 02A 12
132 02B 10
133 02C 0D  . WORD      0D,0A,06,02
134 02D 0A
135 02E 06
136 02F 02

```

```

125
126          ; THE FOLLOWING SUBROUTINE ADJUSTS THE VALUE OF A BASE 2
127          ; LOGARITHM FOUND BY STRAIGHT LINE APPROXIMATION. THE
128          ; CORRECTION TERMS ARE TAKEN FROM THE TABLE ABOVE. THE
129          ; SUBROUTINE HAS 2 ENTRY POINTS:
130
131          .
132          .
133          .
134          .
135          .
136          .
137          .
138          .
139          .
140          .
141          .
142 030 32  AADJ:  RC          ; C = 0 FOR ANTILOG
143 031 F3  JP          ; CONVERSION.
144 032 22  LADJ:  SC          ; C = FOR LOG2 ADJ.
145 033 05  $LD     LD          ; MOVE ADDRESS POINTER BACK
146 034 07  XDS     XDS         ; ONE LOCATION.
147 035 05  LD          ; LOAD CONTENTS OF HI MANTISSA
148 036 37  XDS     XDS         ; AND STORE IT IN THE LO ORDER
149 037 06  X          ; OF THE TEMP MEMORY LOCATION.
150 038 00  CLRA    CLRA        ; SET TABLE POINTER
151 039 52  AISC    TBL         ; (ACC) TO TABLE ADDRESS.

```

COP CROSS ASSEMBLER PAGE: 5  
LOGS

```

152 03A BF          LQID          ; LOAD CORRECTION VALUE TO Q.
153 03B 332C       $GTM:  CQMA    ; TRANSFER Q REGISTER
154 03D 04          XIS          ; CONTENTS TO MEMORY.
155 03F 07          XDS          ;
156 03F 20          SKC          ; ANTILOG?
157 040 80          JSRP         ; YES — COMPLIMENT.
158 041 98          $ADD:  JSRP    ADRO ; ADD CORRECTION VALUE
159
160 042 35          LD          03     ; TO MANTISSA.
161 043 48          $LST:  RET      ; SET POINTER TO
162
163          ; CHARACTERISTIC AND
164          ; RETURN.
165
166          ; 2 ROUTINES ARE CALLED FROM THE SUBROUTINE PAGE BY THIS
167          ; PROGRAM: COMP, ADRO

```

```

167          0020          V1 = TPLS&OFF
168          0002          TBL = V1/16
169
170
171

```

Figure 13.



## Subroutines Used by the Log and Antilog Programs

COP CROSS ASSEMBLER PAGE: 6

LOGS

```

172                                     . FORM
173      0080      . PAGE 02                ; ----- SUBROUTINES ----- ;
174
175      ; THE FOLLOWING ROUTINES RESIDE ON THE SUBROUTINE PAGE. THEY
176      ; ARE CALLED BY THE LOGS PROGRAM BUT ARE GENERAL PURPOSE IN
177      ; NATURE AND FUNCTION AS UTILITY ROUTINES.
178
179
180
181      ; ----- COMPLEMENT 8 BITS ----- ;
182
183      . LOCAL
184
185      ; THIS ROUTINE FORMS IN MEMORY THE 2'S COMPLEMENT OF THE TWO
186      ; ADJACENT DIGITS IDENTIFIED BY THE ADDRESS POINTER. THE
187      ; CONTENTS OF THE ADDRESS POINTER ARE NOT ALTERED.
188
189      ; THERE ARE TWO ENTRY POINTS:
190      ;
191      ; COP: COMPLEMENT 8 BITS.
192      ;
193      ; CMPE: EXTEND THE COMPLEMENT TO AN ADDITIONAL 8 BITS
194      ;
195
196      080      22      COMP:      SC
197      081      00      CMPE:      CLRA                ; SET MINUEND = 0
198      082      06                X                ; AND STORE IN MEMORY.
199      083      10                CASC
200      084      44                NOP                ;
201      085      04                XIS                ;
202      086      00                CLRA                ; SET MINUEND = 0
203      087      06                X                ; AND STORE IN MEMORY.
204      083      10                CASC                ;
205      089      44                NOP                ;
206      08A      04                XIS                ;
207      08B      44                NOP                ; AVOID SKIP IF DIGIT 15.
208      08C      A4                JP          SDB2      ; RETURN THRU SDB2
209                                     ; TO RESTORE POINTER.
210
211
212
213      ; ----- ADD 8 BITS IN ADJACENT REGISTERS ----- ;
214
215      . LOCAL
216
217
218
219      ; THIS ROUTINE ADDS TWO BINARY DIGITS (8 BITS) FROM ANY REGISTER
220      ; TO THE CORRESPONDING TWO BINARY DIGITS IN EITHER REGISTER
221      ; IMMEDIATELY ADJACENT. THERE ARE THREE ENTRY POINTS:
222      ;
223      ; LADR: — RESET CARRY AND ADD 2 DIGIT PAIRS

```

```

224 ; LADD: — ADD 2 DIGIT PAIRS WITH UNMODIFIED CARRY
225 ; ADD1: — ADD 2 SINGLE DIGITS WITH UNMODIFIED CARRY
226
227
228
229
230 08D 32 LADR: RC ; RESET CARRY PRIOR TO ADD.
231 08E 15 LADD: :D 01 ; LD ADDEND AND MOVE TO ADJ REG
232 08F 30 ASC ; ADD AUGEND.
233 090 44 NOP ; AVOID CARRY!
234 091 14 XIS 01 ; STORE SUM AND MOVE TO ADDEND
235 092 15 ADD1: LD 01 ; REPEAT PROCESS
236 093 30 ASC ; FOR
237 094 44 NOP ; HIGH ORDER
238 095 14 XIS 01 ; DIGIT.
239 096 44 NOP ; AVOID SKIP IF DIGIT 15.
240 097 48 $LST: RET ; FINISHED — RETURN!!!!

```

```

241
242
243
244
245 ; ----- ADD 8 BITS IN OPPOSITE REGISTERS ----- ;
246
247 . LOCAL
248
249
250

```

```

251 ; THIS ROUTINE ADDS TWO BINARY DIGITS (8BITS) FROM ANY REGISTER
252 ; TO THE CORRESPONDING TWO BINARY DIGITS IN EITHER REGISTER
253 ; DIRECTLY OPPOSITE. THERE ARE THREE ENTRY POINTS:
254 ;
255 ; ADR0: — RESET CARRY AND ADD 2 DIGIT PAIRS
256 ; ADD0: — ADD 2 DIGIT PAIRS WITH UNMODIFIED CARRY
257 ; ADD1: — ADD 2 SINGLE DIGITS WITH UNMODIFIED CARRY
258
259
260

```

```

261
262 098 32 ADR0: RC ; RESET CARRY PRIOR TO ADD.
263 099 35 ADD0: LD 03 ; LD ADDEND AND MOVE TO OPP REG
264 09A 30 ASC ; ADD AUGEND.
265 09B 44 NOP ; AVOID CARRY!
266 09C 34 XIS 03 ; STORE SUM AND MOVE TO ADDEND.
267 09D 15 AD01: LD 01 ; REPEAT PROCESS
268 09E 30 ASC ; FOR
269 09F 44 NOP ; HIGH ORDER
270 0A0 34 XIS 03 ; DIGIT.
271 0A1 44 NOP ; AVOID SKIP IF DIGIT 15.
272 0A2 48 $LST: RET ; FINISHED — RETURN!!!!

```

```

273
274
275 ; ----- SET DIGIT ADDRESS BACK TWO ----- ;
276
277

```

COP CROSS ASSEMBLER PAGE: 8  
LOGS

```

278             . LOCAL
279
280             ; THIS ROUTINE SUBTRACTS 2 FROM THE CONTENTS OF THE
281             ; DIGIT POINTER (B REGISTER), THE CONTENTS OF THE
282             ; ACCUMULATOR ARE LOST IN THE PROCESS. THE USE OF
283             ; SDB2 ALLOWS ADDRESSING WITHIN THE LOGS SUB
284             ; ROUTINE TO BE RELATIVE TO THE CONTENTS OF THE
285             ; ADDRESS POINTER (B REGISTER) UPON ENTRY.
286             ; SDB2 IS COMMONLY USED IN BYTE OPERATIONS TO RESTORE THE
287             ; DIGIT POINTER TO THE LOW ORDER POSITION.
288             ; THERE ARE TWO ENTRY POINTS:
289
290             ; SDR2:          SET DIGIT ADDRESS BACK 2 AND MOVE TO OPPOSITE REGISTER.
291             ;
292             ; SDB2: SET DIGIT ADDRESS BACK 2 RETAINING PRESENT REGISTER.
293
294
295
296 0A3 35      SDR2:    LD          03          ; MOVE TO OPPOSITE REGISTER.
297 0A4 4E      SDB2:    CBA          ; PLACE DIGIT COUNT IN ACC.
298 0A5 5E          AISC        -2          ; SUBTRACT 2.
299 0A6 44          NOP          ; SHOULD ALWAYS SKIP.
300 0A7 50          CAB          ; PUT DIGIT COUNT BACK.
301 0A8 48          RET          ; FINISHED — RETURN!!
302
303

```

;----- SHIFT LEFT, -----;

```

304             . LOCAL
305
306             ; THIS ROUTINE SHIFTS LEFT THE CONTENTS OF TWO MEMORY
307             ; LOCATIONS ONE BIT. THERE ARE THREE ENTRY POINTS:
308
309             ;
310             ;
311             ; SHLR: RESETS THE CARRY BEFORE SHIFTING
312             ; IN ORDER TO FILL THE LOW ORDER
313             ; BIT POSITION WITH A 0.
314             ;
315             ;
316             ; SHLC: SHIFTS THE STATE OF THE CARRY INTO
317             ; THE LOW ORDER BIT POSITION.
318             ;
319             ; SHL1: SHIFTS LEFT THE CONTENTS OF ONLY
320             ; ONE MEMORY LOCATION. THE STATE
321             ; OF THE CARRY IS SHIFTED INTO THE
322             ; LOW ORDER POSITION OF MEMORY.
323
324
325 0A9 32      SHLR:    RC          ; CLEAR CARRY PRIOR TO SHIFT.
326 0AA 05      SHLC:    LD          ; LOAD FIRST MEM DIGIT.
327 0AB 30          ASC          ; DOUBLE IT.
328 0AC 44          NOP          ; AVOID SKIP.
329 0AD 04          XIS          ; STORE SHIFTED DIGIT.
330 0AE 05      SHL1:    LD          ; LOAD NEXT MEM DIGIT.
331 0AF 30          ASC          ; DOUBLE IT TOO.

```

COP CROSS ASSEMBLER PAGE: 9  
LOGS

```

332 0B0 44          NOP          ; AVOID SKIP, IF ANY
333 0B1 04          XIS          ; STORE SHIFTED DIGIT.
334 0B2 48      $LST:    RET          ; FINISHED — RETURN!
335
336
337             . END

```

# Use of Macro-Assembled Code

National Semiconductor  
COP Brief 3  
May 1980



## Introduction

The use of macro assembled code in a COP400 series program can be beneficial to the user if implemented correctly. Care must be taken to insure that ROM space is not being utilized in a wasteful manner. In many cases a block of commonly used code would lend itself to a subroutine rather than repeating a macro. The purpose of this brief is to illustrate the advantages of the macro capability of the COP400 Product Development System (PDS). Due to modifications in the assembler program there is erroneous information concerning macro calls in the *COP400 PDS Manual*. These modifications are discussed in the section labeled GENERAL.

By using macros the programming process becomes much more general in nature. In some circumstances, with a good macro library, a pseudo higher level language can be created. This higher level of instructions inefficiently utilizes ROM space. However, if the ROM space is available, macros can ease the task of programming. A feasible approach to organized programming might be to work from a macro library and in the event of limited ROM space, optimize code by replacing the macros which are repeatedly used, by a single subroutine and calling statements.

Macros also may be used as programming aids which ease the understanding of the instruction set. When utilizing macros to rename single instructions no ROM space is wasted. Macro statements must be declared at the beginning of a source file. However, this does not utilize ROM space unless the macro is called within the source. Various methods of creating multiple and single instructions macros are discussed below.

## Creating Instruction Macros

One very basic use of macros is to rename instructions or groups of instructions to suit individual preferences. In the example shown the user must add the macro to the source file and each time the new mnemonic is encountered the assembler will create the correct code.

```
B1 = 0           ; EQUATE STATEMENTS
B2 = 0           ; USED FOR PROGRAMMING
B4 = 2           ; CLARITY
B8 = 3           ;
```

```
.MACRO   SZ, BIT
SKMBZ   BIT
.ENDM
```

The renamed instruction may now be utilized in the following way:

```
SZ           B8

OR

SZ           3
```

In both cases 'SKMBZ 3' will be assembled.

By utilizing the equate capabilities the user can even further personalize the instruction set. In the above example 'B1' is equated to '0', 'B2' to '1', etc. This translates a bit position '0,1,2,3' to a bit weight of '1,2,4,8' which may be of preference to the programmer. In any case, the ability to manipulate the instruction set is available to the user without direct modification to the assembler program.

Conditional assembly in conjunction with macro capabilities may be utilized to further ease programming. In the following example the 'JSR' and 'JSRP' instructions are replaced with a simple 'CALL' statement. It is important to allocate the proper number of ROM spaces during pass 1 of the assembler so as to assign a ROM location to correspond to each label. It is not until pass 2 of the assembler that information of label addresses is known. Because of this the macro must be able to determine whether the 'CALL' is a one or two byte instruction. This can be accomplished by use of conditional assembly statements. In the example shown, all subroutines located in page 2 must be labeled by an 'A' followed by the subroutine name. Conversely, subroutines not located in page 2 must not begin with the letter 'A'. Note that the character 'A' was chosen arbitrarily and may be modified to any legal character or characters.

```
.MACRO CALL,X,Y           ; MACRO TO RENAME JSR, JSRP
IFC #1 EQ A              ; TEST IF LABEL IS PREFACED
                          ; BY AN 'A'
JSRP X*Y                 ; YES, ASSEMBLE SINGLE BYTE
.ELSE
JSR X*Y                   ; NO, ASSEMBLE DOUBLE BYTE
.ENDIF                   ; MUST TERMINATE .IF
.endm                    ; TERMINATE MACRO
```

```
CALL      AINC           ; CALL SUB IN PAGE 2
```

This statement will generate:

```
JSRP      AINC
```

AINC must be located in page 2 or an assembler error will occur.

```
CALL SUB           ; CALL SUB NOT IN PAGE 2
```

This statement will generate:

```
JSR      SUB
```

## Macros of Interest

### Table Look-Up Macro

This macro will place the look-up table in the ROM space designated by the LOC parameter or if the parameter is not specified the table will follow in successive locations after being called.

```

MACRO   TABLE,LOC           ; SEG TABLE LOOKUP
.IFC #>0                       ; TEST IF PARAMETER IS THERE
.X' LOC                       ; YES, USE IT
.ELSE                               ; NO, ELIMINATE ROM POINTER
.ENDIF                             ; TERMINATE .IFC
.WORD 0FD                       ; 0
.WORD 061                       ; 1
.WORD 0DB                       ; 2
.WORD 0F3                       ; 3
.WORD 067                       ; 4
.WORD 0B7                       ; 5
.WORD 03F                       ; 6
.WORD 0E1                       ; 7
.WORD 0FF                       ; 8
.WORD 0E7                       ; 9
.WORD 0CF                       ; P
.WORD 0EF                       ; A
.WORD 07D                       ; U
.WORD 09D                       ; C
.WORD 08F                       ; F
.WORD 000                       ; BLANK
.ENDM

TABLE 024                       ; SET ROM POINTER AT ROM
                                       ; LOCATION 024<hex>

OR

TABLE                               ; START SEVEN SEG AT PRESENT
                                       ; ROM LOCATION

```

The code generated will correspond to the look-up table given in the macro. This table may be modified to suit any particular symbol. Sixteen segment arrays are listed only to take advantage of the LQID instruction. These may be modified to the user's preference.

Additional Macro information is available in the *COP400 Product Development System Manual*.

### General

The *COP PDS Manual* defines parameter delimiters when using macros as commas or blanks. When creating the macro, parameters must be separated by commas whereas blanks are not acceptable. When calling the macro it is acceptable to delimit the parameters by either blanks or commas.

In order to assure correct assembly when using the .IFC or .IFC directives it is essential to terminate these directives by a .ENDIF. This point is not emphasized in the manual. However it is important in the assembly process.

The .LIST directive may be used to suppress the macro listing in the source or to expand it. The *COP PDS Manual* covers LIST options in detail.



## L-Bus Considerations

Users of the COP400 family of microcontrollers should be aware that certain outputs exhibit peculiarities that preclude their use as clocks for edge sensitive devices such as flip-flops, counters, shift registers, etc. All family members excluding the COP410L and COP411L may

generate false states on L<sub>0</sub>-L<sub>7</sub> during the execution of the CAMQ instruction. Figure 1 contains a short program to illustrate this.

In this program the internal Q register is enabled onto the L lines and a steady bit pattern of logic highs is output on L<sub>0</sub>, L<sub>1</sub>, L<sub>6</sub>, L<sub>7</sub>, and logic lows on L<sub>2</sub>-L<sub>5</sub> via the two-byte CAMQ instruction. Timing constraints on the device are such that the Q register may be temporarily loaded with the second byte of the CAMQ opcode (X'3C) prior to receiving the valid data pattern. If this occurs, the opcode will ripple onto the L lines and cause negative-going glitches on L<sub>0</sub>, L<sub>1</sub>, L<sub>6</sub>, L<sub>7</sub>, and positive glitches on L<sub>2</sub>-L<sub>5</sub>. Glitch durations are under 2 microseconds, although the exact value may vary due to data patterns, processing parameters, and L line loading. These false states are peculiar only to the CAMQ instruction and the L lines. The user should experience no difficulty interfacing with other COP420 outputs such as G<sub>0</sub>-G<sub>3</sub> and D<sub>0</sub>-D<sub>3</sub> to edge sensitive components.

```
START:
  CLRA          ; ENABLE THE Q
  LEI    4      ; REGISTER TO L LINES
  LBI    TEST
  STII   3
  AISC   12

LOOP:
  LBI    TEST          ; LOAD Q WITH X'3C
  CAMQ
  JP     LOOP
```

Figure 1. Glitch Test Program

# Software and Opcode Differences in the COP444L Instruction Set

National Semiconductor  
COP Brief 5  
May 1980



The COP444L is essentially a COP420L with double RAM and ROM. Because of this increased memory space certain instructions have expanded capability in the COP444L. Note that there are no new instructions in the COP444L and that all instructions perform the same operations in the COP444L as they did in the COP420L. The expanded capability is merely to allow appropriate handling of the increased memory space. The affected instructions are:

JMP a (a = address)  
JSR a (a = address)  
LDD r,d (r,d = RAM address Br,Bd)  
XAD r,d (r,d = RAM address Br,Bd)  
LBI r,d (r,d = RAM address Br,Bd; only two byte form of the instruction affected)  
XABR

The JMP and JSR instructions are modified in that the address a may be anywhere within the 2048 words of ROM space. The opcodes are as follows:

JMP	<table border="1"><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>a<sub>10:9:8</sub></td></tr></table>	0	1	1	0	0	a <sub>10:9:8</sub>	JSR	<table border="1"><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>a<sub>10:9:8</sub></td></tr></table>	0	1	1	0	1	a <sub>10:9:8</sub>
0	1	1	0	0	a <sub>10:9:8</sub>										
0	1	1	0	1	a <sub>10:9:8</sub>										
	<table border="1"><tr><td>a<sub>7:0</sub></td></tr></table>	a <sub>7:0</sub>		<table border="1"><tr><td>a<sub>7:0</sub></td></tr></table>	a <sub>7:0</sub>										
a <sub>7:0</sub>															
a <sub>7:0</sub>															

The LDD, XAD, and two byte LBI are modified so that they may address the entire RAM space. The opcodes are as follows:

LDD	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	1	0	0	0	1	1	XAD	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1												
0	0	1	0	0	0	1	1												
	<table border="1"><tr><td>0</td><td>r</td><td>d</td></tr></table>	0	r	d		<table border="1"><tr><td>1</td><td>r</td><td>d</td></tr></table>	1	r	d										
0	r	d																	
1	r	d																	
LBI	<table border="1"><tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	0	0	1	1	0	0	1	1										
0	0	1	1	0	0	1	1												
	<table border="1"><tr><td>1</td><td>r</td><td>d</td></tr></table>	1	r	d															
1	r	d																	

The XABR instruction change is transparent to the user. The opcode is not changed nor is the function of the instruction. The change is that values of 0 through 7 in A will address registers in the COP444L — i.e. the lower three bits of A become the Br value following the instruction. In the COP420L, the lower two bits of A became the Br value following an XABR instruction.

Note that those instructions which have an exclusive-or argument (LD, X, XIS, XDS) are not affected. The argument is still two bits of the opcode. This means that the exclusive-or aspect of these instructions works within blocks of four registers. It is not possible to toggle Br from a value between 0 and 3 to a value between 4 and 7 by means of these instructions.

There are no other software or opcode differences between the COP444L and the COP420L. Examination of the above changes indicates that the existing opcodes for those instructions have merely been extended. There is no fundamental change.

# RAM Keep-Alive

National Semiconductor  
COP Brief 6  
May 1980



COP Brief 6

A COPS™ application is a small scale computer system and the design of a power shut-down is not trivial. During the time that power is available, but out of the designed operating range, the system must be prevented from doing anything to harm protected data. This will typically involve some type of external protection or timing circuit.

There is an option on the COP420, 420L, and 410L parts called "RAM Keep-Alive" that provides a separate power supply to the RAM area of the chip via the CKO pin. The application of power to the RAM while the remainder of the chip has been powered down via  $V_{CC}$  will keep the RAM "alive".

However, the integrity of data in the RAM is not only a function of power but is also influenced by transient conditions as power is removed and reapplied. During power-on, the Power On Reset (POR) circuit will keep transients from causing changes in the RAM states. The condition of power loss will have some probability of data change if external control is not used.

At some point below the minimum operating voltage certain gates will no longer respond properly while others may still be functional until a much lower voltage. During this transition time any false signal could cause a false write to one or more cells. Another effect could be to turn on multiple address select lines causing data destruction.

Testing the rate of data change is very difficult because it must be done on a statistical basis with many turn/on-turn/off cycles. Two factors have a major bearing on the numbers derived by testing. One is to call any change in a related data block a failure, even though more than one bit in that block may have changed (this latter case may well be due to the "address select mode"). The second factor is that without massive instrumentation it is impossible to examine the data after each power cycle. Indeed, to do so might have caused errors!

By running the power cycle for a period of time and then looking for changes, one could overlook multiple changes thus reducing the error rate. This has been minimized by more frequent checking which indicates that the errors are spread out randomly over time.

With a power supply that drops from 4.5 to 2V in approximately 100ms, the drop-out rate is 1 in 5k to 6k power cycles. Reducing the voltage fall time will cause an improvement in the number of cycles per drop-out. This will reach a limit condition of a very high number (1 per 1 million?) when the power falls within one instruction cycle (4-10 $\mu$ s for the 420, 15-40 $\mu$ s for the "L" parts). Attaining very rapid fall time may cause problems due to the lack of decoupling/bypass capacitance. By inserting an electronic switch between the regulator and  $V_{CC}$  of the COP chip one might be able to meet this type of fall time. By implication some type of sensing is required to cause the switching.

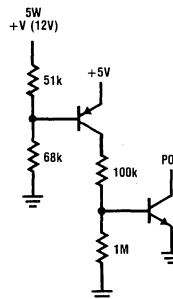
The desirable approach is to force the COP reset input to zero before the voltage falls below 4.5V. This provides a drop out rate of approximately 1 in 50k for the "L" parts and 1 in 100k for the 420. By also stopping the clock of the "L" parts they can achieve a drop-out rate similar to the 420. While not perfect, the number of cycles between data error should be considered with respect to the needs of the application.

The external circuitry to control the chip during the power transition has several implementations each one being a function of the application. The simplest hardware is found in a battery powered (automotive) application. The circuit must sense that the switched 12V is falling (e.g., at some value much below 12V and still greater than 5V). This can be done by using the unswitched 12V as a reference for a divider to a nominal voltage of 8V. As the switched 12V drops below the reference a detector will turn on a clamp transistor to a series switch, the POR, and/or the clock circuit (Figure 1). It should be noted that this draws current during the absence of the switched 12V circuit.

In non-automotive usage a similar circuit can be used where there is a stable reference voltage available to use with the comparator/clamp. Thus a 3.6V rechargeable Ni-Cad battery could be used as the reference voltage and  $V_{RAM}$  if the appropriate divider is used to level shift to this operating range.

In AC line-powered applications, a similar method could be used with the raw DC being sensed for drop. Another method would be to sense that the line had missed 2-3 cycles either by means of a charge pump or peak detection technique. This will provide the signal to turn on the clamp. One must make this faster than the time to discharge the output capacitance of the power supply, thus assuring that the clamp has performed its function before the supply falls below spec value.

In conclusion, to protect the data stored in RAM during a power-off cycle, the POR should go low before the  $V_{CC}$  power drops below spec and come up after  $V_{CC}$  is within spec. The first item must be handled with an external circuit like Figure 1 and the latter by an RC per the data sheet.





# MICROBUS™ Programming Considerations

National Semiconductor  
COP Brief 7  
May 1980



## Introduction

The COP402 MICROBUS™ is a peripheral microprocessor device and its operating characteristics are described in the COP402M data sheet and the *Chip User's Manual*. Given in this brief are some clarifications as to the allowable option selection and also as to programming requirements that are not readily obvious.

## COPS IN Input Port Options on the COP402M

In the COP402M configuration,  $IN_0$  is a general purpose latched input with a load device to  $V_{CC}$ . All other IN inputs (CS, RD, and WR), are selected as high impedance inputs without pull-up devices.

The COP402M and the COP420M will execute ININ and INIL instructions.  $IN_0$  information will be latched in accordance with the criteria specified in the data sheet (min. 2 inst. cycle time at logic zero), as will the WR, ( $IN_3$ ) input if these criteria are met. If the WR pulse does not meet the 2 instruction cycle criteria, yet does satisfy MICROBUS timing, the status of the IL latch corresponding to the WR input ( $IN_3$ ) cannot be predicted when the status of the IL latches is read in via an INIL instruction.

When executing the ININ instruction, the status of  $IN_0$  and the MICROBUS signals will be read in with the exception of the RD ( $IN_1$ ) signal. This signal will always read in as a logical one.

## COPS IN Input Port Options on the COP420M

When selecting a MICROBUS option it is possible to select either load devices to  $V_{CC}$  or high impedance inputs on  $IN_0$  and all MICROBUS signals. These options may be chosen individually corresponding to  $IN_0$ , CS, WR, and RD signals. There is also a choice between standard TTL input levels or a High Trip option for the IN and MICROBUS inputs. The only restriction (for all 400 series devices) is that when either a High Trip or TTL trip levels are chosen, they must be selected in blocks corresponding to that input port. For example, all IN lines must have High Trip, rather than just one IN line.

## MICROBUS™ Programming Considerations

The COP402M data sheet describes the handshaking protocol required when implementing the COP420M as

a microprocessor peripheral device. When a WR strobe is detected, an internal reset of the  $G_0$  latch occurs. This signal indicates that data is ready to be transferred to the Q latches from the microprocessor bus. Due to the relatively short timing requirements on the WR strobe signal it is necessary to latch the write request such that under program control the COP device can service the write request. Upon completion of the data transfer and any task that may have been performed, the user then signals the microprocessor that it is available once again by setting the  $G_0$  latch. This portion of the handshaking (setting  $G_0$ ) is the only time that the G Port should be used as an output port. All G Ports in the MICROBUS configuration should be used only as input in order to guarantee that a WR strobe is not missed. When using the G Port as an output Port it is possible that a WR pulse may be ignored as explained in the example below. The G Port may be utilized as an output port in the following way, however, there is a 3 cycle period that if a WR pulse occurred it would be ignored.

GPIN:	LBI	RAM	:	POINT TO RAM LOCATION
	ING		:	READ THE G PORT
	X		:	STORE IN RAM
	SMB	X	:	CHANGE G PORT INFO TO BE SENT OUT
	SKGBZ	O	:	SEE IF WR STROBE HAS OCCURRED
	JP	OUT	:	HAVE NOT BEEN INTERRUPTED (YET)
	JP	SERVICE	:	GO SERVICE WR REQUEST
OUT:	OMG		:	OUTPUT NEW G PORT INFORMATION

If a write pulse occurred during the JP to OUT or the OMG instructions it would not be recognized because the OMG will set the  $G_0$  latch to a logic one, signalling to the microprocessor that the WR strobe has been serviced.

It is possible to output to the G Port after WR and before  $G_0$  is set, and not miss a WR request. This means that the data outputted on the G lines will be updated only after the microprocessor has initiated an interrupt.

## General

The COP402M data sheet specified all IP address lines as TTL compatible, with a fan out of one. Address lines IP4 and IP5 do not meet this criterion, although all other IP lines do. It is sufficient to say that all IP lines are LSTTL compatible with a fan out of one, the restricting factor being IP4 and IP5, ( $I_{OL} @ 0.4V, 360\mu A = I_{OH} @ 3.0V = 50\mu A$ .)



## COPS Peripheral Chips

There are several I/O peripheral chips that are compatible with the COPS microcontrollers by communicating through the serial I/O port. Table 1 shows a listing of those circuits. Two different sets of timing employed by them are shown in Figure 1. A brief description of the electrical characteristics of each chip is given below.

### COP452 Frequency and Counter Chip

The COP452 frequency and counter chip is fabricated by N-channel silicon gate process. The chip operates between 4.5V and 9.5V. It contains a TRI-STATE™ output to be connected to the SI pin of the COPS controller. This output can drive the SI pin of a standard or a low power COPS controller provided that standard TTL input level option is chosen for the SI pin. If the higher input level option is chosen, or a CMOS COPS controller is used, an external resistor may be used to increase the HIGH output level. The LOW level will also increase.

### COP470 V.F. Display Driver

The COP470 V.F. display driver is fabricated by a PMOS process. It operates between 4.5V and 9.5V with a high voltage supply pin for output drivers to drive fluorescent displays. The input levels on this chip are different from other chips. The LOW level is between 0V and  $V_{CC} - 4V$ , and the HIGH level is between  $V_{CC} - 1.5V$  to  $V_{CC}$ . The input LOW level will be between 0V and 0.5V when  $V_{CC}$  is 4.5V. If  $V_{CC}$  is above 5V, the input HIGH level will be above the CMOS input HIGH level, e.g., with  $V_{CC}$  being 9.5V, the minimum input HIGH level will be 8V, compared to 6.8V for CMOS minimum input HIGH level. The COPS controller data sheet will not accurately show the propagation delay. To obtain a conservative estimate of the propagation delay, assume that delay comes from R-C charging time, with the capacitance and time necessary to charge to  $0.7V_{CC}$  given in the data sheet (COPS to CMOS interface), extrapolate the time to the minimum HIGH level for that power supply voltage. This value should be a good conservative estimate.

### COP472 LCD Driver

The COP472 LCD driver is fabricated by a low voltage CMOS process. The driver operates between 3V and 5.5V. The clock (SK), data input (DI), and chip enable (CE) may tolerate a 10V signal. The actual power supply used will depend on the operating voltage of the LCD.

### COP498 Read/Write Memory and Timer Chip

The COP498 read/write memory and timer chip is fabricated by a low voltage CMOS process. The chip operates between 2.5V and 5.5V. Some I/O, including clock (SK), data input (DI), and chip enable (CE) may tolerate a 10V signal. When interfacing to a COPS controller with a

higher power supply, data output (DO) should not rise above the COP 498 power supply.

### DS8906 PLL Chip

DS8906 PLL chip is fabricated by a  $i^2L$  process. The chip operates between 4.75V and 5.25V. The inputs may tolerate a 9V signal. The maximum input source current is  $10\mu A$  and the maximum input sink current is  $25\mu A$ .

### MM5450 LED Display Driver

The MM5450 LED display driver is fabricated by an N-channel metal gate process. The chip operates between 4.75V and 11V.

### TTL SSI/MSI/LSI Interface

The 7400 series logic operates between 4.75 and 5.25V only. The standard and CMOS COPS controller outputs can directly drive one input and maintain the TTL valid input levels. If it is also necessary to drive CMOS or PMOS in a 5V system, buffers or an external 4.7k pull-up resistor may be added. This resistor together with a TTL load may increase the maximum output LOW level to 0.5V. If a TTL output needs to drive a CMOS COPS controller input or a standard COPS controller input with a high input option from a TTL buffer, a TTL to MOS buffer or an external pull-up 4.7k resistor may be added.

### LSTTL SSI/MSI/LSI Interface

The 74LS series logic operates between 4.75V and 5.25V only. The standard and CMOS COPS controller outputs can directly drive four inputs and maintain the LSTTL valid input levels. If it is necessary to drive also CMOS or PMOS circuits in a 5V system, buffers or a 4.7k pull-up resistor may be added. This resistor together with four LSTTL loads may increase the maximum output LOW level to 0.5V. If it is necessary to drive a CMOS COPS controller input or the standard COPS controller input with a high input option from an LSTTL output, a TTL to MOS buffer or an external 4.7k pull-up resistor may be added.

The low-power COPS controller outputs can directly drive one LSTTL input and maintain the valid LSTTL input levels. If it is also necessary to drive CMOS or PMOS circuits in a 5V system, buffers or a 22k resistor may be added. This resistor together with the LSTTL load will maintain a maximum output LOW level of 0.3V at the serial out (SO) or clock (SK) outputs. If it is necessary to drive a low power COPS controller input with a high input level option from LSTTL output, a TTL to MOS buffer or an external 22k pull-up resistor may be added.

Table 1. COPS Compatible Peripheral Chips

Peripheral Chips	Process	V <sub>CC</sub> (V)	DI/SK		CE Polarity	DI Setup Time (μs)	Set Frequency	
			Max. LOW (V)	Min. HIGH (V)			Min. (kHz)	Max. (kHz)
COP452	NMOS	4.5-9.5	0.8	2.0	-	1.0	24	265
COP470	PMOS	4.5-9.5	V <sub>CC</sub> - 4	V <sub>CC</sub> - 1.5	-	1.0	0	265
COP472	CMOS	3.0-5.5	0.3V <sub>CC</sub>	0.7V <sub>CC</sub>	-	1.0	0	265
COP498	CMOS	2.5-5.5	0.3V <sub>CC</sub>	0.7V <sub>CC</sub>	+	0.3	24	265
DS8906	I <sup>2</sup> L	4.75-5.25	0.8	2.0	-	0.3	0	625
MM5450	NMOS	4.75-11.0	0.8	2.0	-	0.3	0	500

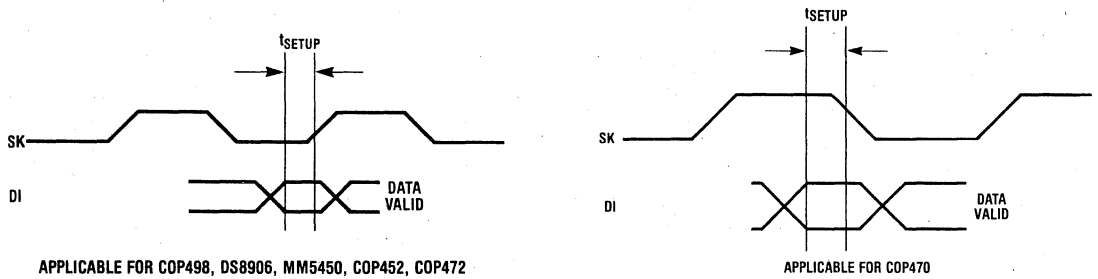


Figure 1. Serial Input Data Timing



# Serial Interface Between COPS™ Microcontrollers and Peripheral Chips

A variety of I/O and data memory expansion chips are available to the COPS™ controllers for different applications. Many of them use the serial port for data transfers, and the COPS controllers allow multiple peripheral chips to be tied in parallel for this purpose (see Figure 1). This paper will discuss the system hardware considerations needed to execute the data transfers. Most COPS controller pins allow various I/O options, and the user should refer to the appropriate data sheet for specific options information. For this discussion, it is assumed that serial input (SI) is a high impedance input for simplicity, and serial output (SO) and clock (SK) are push-pull outputs for lower switching time. All the chips are assumed to have the same power supply. The interface response characteristics may be divided into two parts: static and dynamic.

## I. Static Response

When the output to the serial interface changes state, the input connected to the interface should detect the change. This is done by keeping the output signal level within the specified HIGH or LOW level range of the input. There are two types of transistors used in integrated circuits, namely, MOS and bipolar transistors. They present different equivalent circuits to the output driver and therefore are considered separately.

### 1. MOS (NMOS, CMOS, PMOS)

The MOS inputs look like capacitive loads to these outputs, with a maximum leakage current usually specified. The COPS output driver must be able to sink or source the total maximum leakage current resulting from various inputs connected to it, and keep the signal level within the valid HIGH or LOW value range. Without any leakage, the outputs should reach the same level as that achieved when the output is not loaded.

Different IC devices have different HIGH and LOW input ranges. Most NMOS parts have TTL compatible levels for 5V operation, i.e. 0V to 0.8V for LOW level and 2.0V to  $V_{CC}$  for HIGH level. The NMOS COPS controllers also allow a mask-programmed optional range: 0V to 1.2V for LOW level and 3.6V to  $V_{CC}$  for HIGH level. Most CMOS parts allow 0V to 0.3 $V_{CC}$  for LOW level, 0.7 $V_{CC}$  to  $V_{CC}$  for HIGH level. The COP470, a V.F. display controller in PMOS process, has 0V to  $V_{CC} - 4V$  for LOW level, and  $V_{CC} - 1.5V$  to  $V_{CC}$  for HIGH level.

When peripheral chips of different MOSFET types are connected together, the output from the controller must satisfy all the input requirements for each peripheral chip. When peripheral chips with TRI-STATE™ outputs are tied to SI, each of the outputs must satisfy the input level of the COPS controller, while supplying the maximum leakage current to the TRI-STATE outputs. If an input and an output have incompatible levels, external circuits may be necessary for level shifting.

### 2. Bipolar (TTL, LSTTL, I<sup>2</sup>L)

Standard and CMOS COPS controller outputs are designed to drive one TTL load or four LSTTL loads, whereas the low power COPS controller outputs can drive only one LSTTL load. If more drive is necessary, a buffer will be needed. Standard and low power COPS controller inputs have TTL input levels, therefore multiple TTL/LSTTL TRI-STATE outputs can be connected together directly to SI. The maximum total leakage current at the SI input and all the TRI-STATE outputs determine the maximum number of TRI-STATE outputs that can be tied together. The TTL/LSTTL output levels are not compatible with the CMOS COPS input levels so that extra external components will be necessary for the interface. The simplest solution is to use a pull-up resistor to raise the HIGH output level. A disadvantage is that the LOW output level will be increased.

Bipolar integrated circuits in other processes, e.g., a DS8906 PLL chip manufactured by I<sup>2</sup>L process, may have different input levels and different input source and sink requirements. It is necessary to determine whether the COPS output can meet the current requirement and maintain a valid voltage level for the input.

### 3. Mixed (Bipolar and MOS)

Both bipolar and MOS peripheral chips may be used in the same system provided that all the current and voltage requirements are met. Most NMOS and bipolar chips can be mixed together because of similar input voltage levels. CMOS and PMOS chips, on the other hand, cannot be mixed with bipolar chips directly because of the higher HIGH level required. The COPS output HIGH level may be loaded down by the bipolar circuit to an unacceptable HIGH level for the CMOS/PMOS inputs. External circuits will be needed to solve the problem. The simplest solution is a pull-up resistor which improves the source current and raises the output to a higher HIGH level. The resistance should not be too small to increase the LOW level above TTL specification.

## II. Dynamic Response

Provided an output can switch between a HIGH level and a LOW level, it must do so in a predetermined amount of time for the data transfer to occur. Since the transfer is synchronous, the timing is relative to the system clock (provided by SK). For example, if a COPS controller outputs a value at the falling edge of the clock and is latched in by the peripheral device at the rising edge, then the following relationship has to be satisfied:

$$t_{DELAY} + t_{SETUP} \ll t_{CK} \text{ (see Figure 1),}$$

where  $t_{CK}$  is the time from data output starts to switch to data being latched into the peripheral chip,  $t_{SETUP}$  is the setup time for the peripheral device where the data has to be at a valid level, and  $t_{DELAY}$  the time for the output to read the valid level.  $t_{CK}$  is related to the system

clock provided by the SK pin of the COPS controller and can be increased by increasing the COPS instruction cycle time. Maximum  $t_{SETUP}$  is specified in the peripheral chip data sheets. The maximum  $t_{SETUP}$  is specified in the peripheral chip data sheets. The maximum  $t_{DELAY}$  allowed may then be derived from the above relationship.

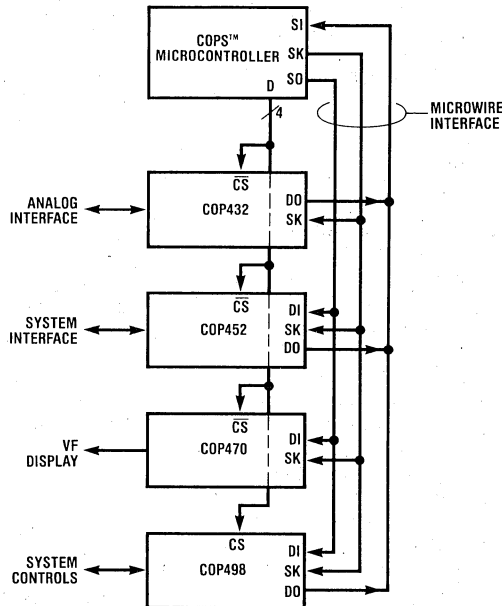
Most of the delay time before the output becomes valid comes from charging the capacitive load connected to the output. Each integrated circuit pin has a maximum load of 7 pF. Other sources come from connecting wires and connection from PC boards. The total capacitive load may then be estimated. The propagation delay values given in data sheets assume particular capacitive loads.

If the calculated load is less than the given load, those values should be used. If the calculated load is greater, a conservative estimate is to assume the delay time is proportional to the capacitive load. The COPS data sheet

provides two sets of values, one for external loads that includes TTL/LSTTL inputs, the other for pure capacitive loads (MOS inputs).

If the capacitive load is too large to satisfy the delay time criterion, then three choices are available. An external buffer may be used to drive the large load. The COPS instruction cycle may be slowed down. An external pull-up resistor may be added to speed up the LOW level to HIGH level transition. The resistor will also increase the output LOW level and increase the HIGH level to LOW level transition time, but the increased time is negligible as long as the output LOW level changes by less than 0.3V. For a 100pF load, the standard COPS controller may use a 4.7k external resistor, with the output LOW level increased by less than 0.2V. For the same load, the low power COPS controller may use a 22k resistor, with the SO and SK output LOW levels increased by less than 0.1V.

This is MICROWIRE™  
(Example System)



# Power Seat with Memory

National Semiconductor  
COP Brief 11  
May 1980



## Introduction

As cars continue to be downsized, more extra features are being offered to the car purchaser to individualize the car to his personal taste. This is especially true with electronic equipment. Automobiles are now available with digitally tuned radios, trip computers, digital gauges and other electronic systems. These have been made possible only recently by the increasing level of semiconductor integration and the resulting lower cost for the components that make up each system.

This article describes another application for electronics in an automobile, a power seat with position memory. This seat features powered adjustment in 8 different directions, the ability to store 2 sets of position information in memory, and instant recall and automatic adjustment to either of the 2 positions. The seat can therefore be adjusted to accommodate 2 different drivers or 2 different driving positions for the same driver and automatically adjust to either of these positions on demand.

## System Description

A block diagram of the seat control system is shown in Figure 1. The heart of the system is the COP420L microcontroller. This part is one of National Semiconductor's COP400 Family of 4-bit, 1-chip microcontrollers. Motor control information is output to the TRI-STATE<sup>®</sup> octal latch and information from the seat sensors is input through the TRI-STATE octal buffer. Manual adjustment of the seat is provided by 8 switches mounted on a console. These manual controls have priority over automatic control via the TRI-STATE control pin on the latch. In addition, the controller software will terminate automatic control if it detects the seat being adjusted in a way different from its programmed positions. This provides for manual override and is necessary as a safety precaution. The system will operate manually even with the controller part removed, which gives a fail-safe operation.

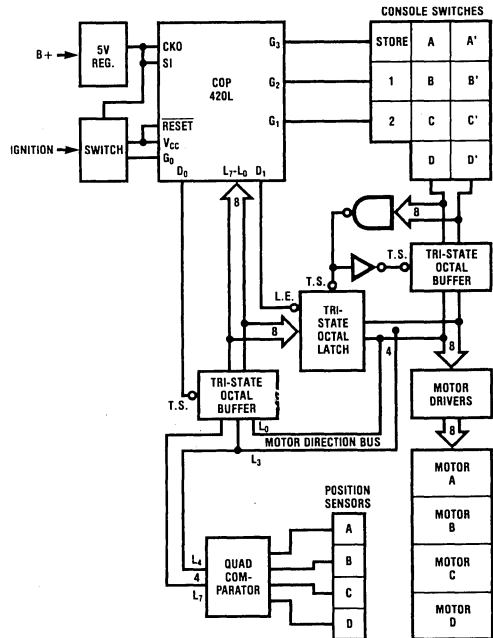


Figure 1. Block Diagram

## The Controller

The COP420L is an N-channel MOS device with 1K $\times$ 8-bit program memory and a 64 $\times$ 4-bit data memory. Its internal architecture is shown in Figure 2, and electrical specifications are shown in Figure 3. In this application, the bidirectional TRI-STATE L lines are used to output motor control information to the motor control latch and also are used to input

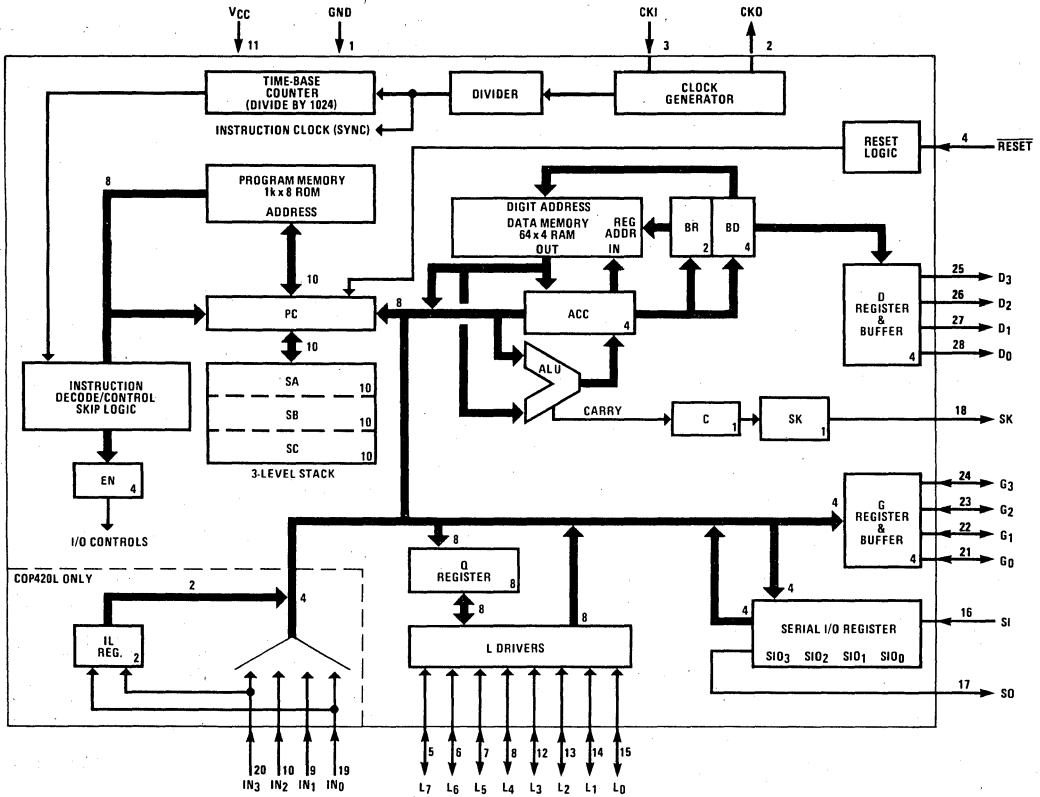


Figure 2. COP420L Block Diagram

Operating Voltage	4.5V-9.5V
Operating Supply Current	8mA (max)
RAM Supply Requirements	3mA (max) @ 3.3V
Minimum Instruction Cycle Time	16µs

Figure 3.

seat position sensor information. The selection of the L lines as inputs or outputs is done through software control and a D<sub>0</sub> line controls the operation of the TRI-STATE buffer to coordinate the reading of sensor information or outputting motor control information. The D<sub>1</sub> line controls the operation of the TRI-STATE latch. The G<sub>1-3</sub> lines are used to detect closure of the memory control keys. Pressing 1 preceded by pressing SET will store the present seat position in memory location 1 and pressing 2 preceded by SET will store position information in memory location 2. Pressing 1 or 2 without first pressing SET will cause the seat to adjust to the respective previously stored position. The remaining G<sub>0</sub> line is used to detect the car's ignition being turned off so the seat can be moved back to allow easy exit from the car.

The IN lines of the COP420L are not used in this design but could be used to interface more memory control keys. There is available space in RAM to store additional seat positions if desired.

The CKO pin is used to provide power to the on-chip RAM in order to retain seat position information when the ignition switch is turned off. Power to the controller and other components is removed in this condition to minimize current drain on the automobile battery.

### System Power Supply

Careful consideration must be given to designing power supply circuitry for automotive electronic systems. Adequate protection must be provided against the electrical transients present in the automotive electrical system. These transients are listed in Figure 4. In addition to these transients, there exists the possibility of 2-battery jumps (+24V) and reversed 2-battery jumps (-24V). All of these must be protected against for reliable operation.

National Semiconductor's LM2930 was specifically designed for supply regulation in automotive electric systems. Its electrical characteristics are listed in

Load Dump	50V $\tau = 200$ ms
Inductive Load Switching	$\pm 250$ V $\tau = 1$ ms
Mutual Coupling	$\pm 450$ V $\tau = 0.1$ $\mu$ s

Figure 4. Automotive Transients

Max Operating Input Voltage	26V
Over-Voltage Protection	40V
Output Voltage	4.5V - 5.5V
Line Regulation	80mV max
Load Regulation	50mV max
Dropout Voltage	0.6V max

Figure 5. LM2930 Specifications

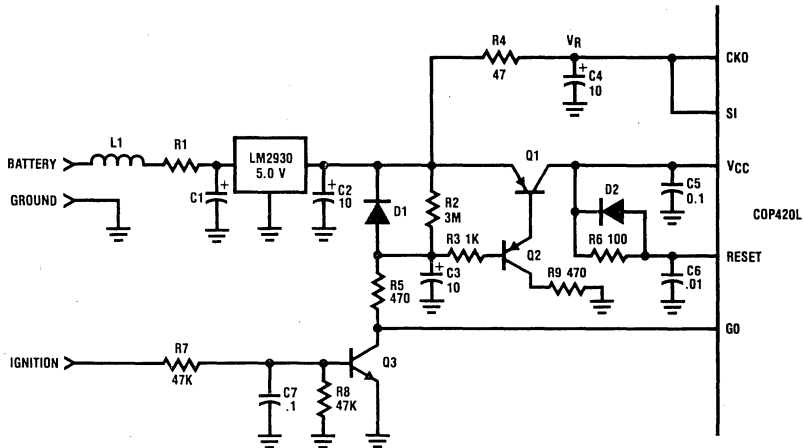


Figure 6. Power Supply Circuitry

Figure 5. This part is internally protected against reverse battery installation and 2-battery jumps. Therefore, all that is needed is to protect the part from input voltages over 40V. This is easily done with an R-L-C circuit. Designing for load dump protection will give protection against the larger but faster transients.

In order to minimize battery drain,  $V_{CC}$  is turned off to all the circuitry except for the COP's RAM when the ignition is turned off. Refer to Figure 6. When the ignition is on, Q3 provides drive to Q1 and Q2. Q1 also holds  $G_0$  low. When the ignition is turned off, the program software detects the low on  $G_0$  being released and performs a routine to park the seat.  $V_{CC}$  is supplied to the controller and circuitry until C3 charges up through R2 to turn off Q1 and Q2, allowing sufficient time for the seat to reach its parked position. Each time  $V_{CC}$  is turned on, the program software checks the contents of the serial register to see if power to the RAM has been lost. If the serial register is all "ones," power has not been lost. If the contents are all "zeros," RAM power has been lost and the RAM and seat are initialized.

This procedure also occurs if the car battery has been disconnected. When it is reconnected, C3 is initially discharged and turns on Q1 and Q2.  $V_R$  is

delayed by R4 and C4 and therefore the serial register is loaded with "zeros" and the RAM and seat are initialized. C3 then charges up and turns off Q1 and Q2 and the system returns to standby. (Note: The values of the timing components have been established experimentally.)

#### System Interface — Output

The 8 different directions of movement of the seat are provided by 4 drive motors. These 8 directions are:

- A — Tilt Seat Back Rearward
- A' — Tilt Seat Back Forward
- B — Move Seat Backward
- B' — Move Seat Forward
- C — Front of the Seat Up
- C' — Front of the Seat Down
- D — Rear of the Seat Up
- D' — Rear of the Seat Down

The motors that move the seat typically draw 2 amps each when running, but draw up to 10 amps each when stalled. The motors also require bidirectional



drive to operate them both in forward and reverse. For these reasons, relays were chosen over semiconductors for the interface.

A high voltage open collector buffer is used to energize the desired relay from the motor control bus. Zener diodes are necessary from the collectors to ground to clamp the inductive turn-off transient to a voltage below the  $V_{CE0}$  of the transistor. These diodes also provide protection for the buffers against load dump and the other transients on the battery supply line.

### System Interface — Input

For the controller to be able to store a seat position in memory and then later to adjust the seat to that position, it is necessary for the controller to know the relative seat location at all times. This is accomplished through sensors mounted on the seat mechanism.

In the prototype, two types of sensors were used. Both types of sensors provided digital information to the controller.

A photodetector package was used with a slotted disc on the seat back. The disc was mounted on the gear mechanism, and as it revolved it interrupted the light source in the detector package as the seat back angle was adjusted. A comparator is used to detect these interruptions and provide logic level compatible pulses to the controller. The controller keeps a running count of these pulses to know where the seat back is at all times. Direction information is fed back to the controller from the motor control bus so the controller knows whether to add or subtract the pulses. This is shown in Figure 1.

The other 3 seat movement mechanisms required a different type of sensor due to their construction. These mechanisms are driven through a flexible cable by a motor. A photodetector sensor could not be added without some major modifications. Therefore, the sensor selected was a speed sensor commonly used for automobile cruise control and could be inserted between the motor and the drive cable. This type of sensor generates an AC waveform that corresponds to the revolutions of the motor. The AC signal is conditioned by a comparator to produce logic level pulses. The sensor is constructed with multiple poles so a divider is used after the comparator to provide the correct number of pulses for the full travel of the seat mechanism.

### An Alternative Approach

Another approach to a seat control system is to use analog sensors instead of digital sensors to track seat position. A block diagram of this approach is shown in Figure 7. The position sensors are potentiometers mounted to the seat mechanism. The multiplexer, under software control, selects which sensor is to be measured and the A-to-D converter inputs the position information to the controller in 8-bit binary format.

It is not necessary in this approach to keep a constant account of the seat's position since it can be determined at any time by polling the potentiometer sensors. The software is therefore much simplified and allows the use of a COP410L which has one-half the memory sizes of the COP420L. The signal conditioning circuitry for the digital sensors that was described earlier is also eliminated. These two things plus the lower cost for potentiometer sensors result in an overall system cost advantage.

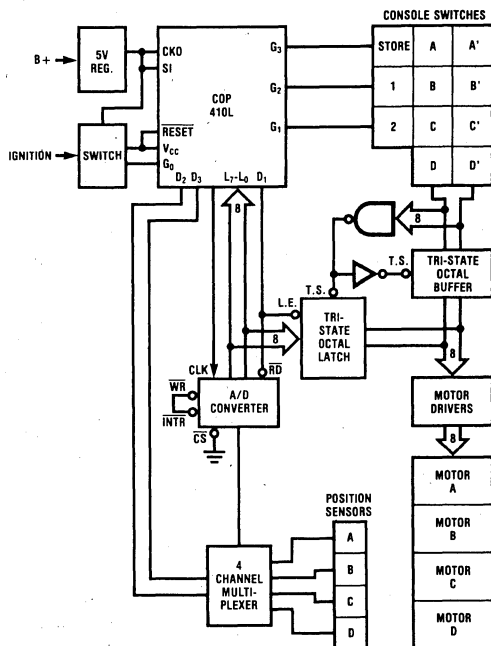


Figure 7. Block Diagram

### Conclusion

A control system for a power seat that has the ability to store and recall preferred driving positions can be designed using a low-cost 4-bit, 1-chip microcontroller and adds to the list of electronic systems being offered today for safety, comfort, and convenience of the automobile driver.

### Acknowledgements

My thanks to Recaro, USA, for supplying the seat for the prototype described in this article and to Jim Troutner, National Semiconductor, for writing the software routines.

TRI-STATE® is a registered trademark of National Semiconductor Corporation.

# An Automotive Diagnostics Display

National Semiconductor  
COP Brief 12  
May 1980



## Introduction

The continued downsizing of the automobile has put a premium on instrument panel space. This has provided the opportunity for electronics to merge the various displays now found in the current automobile to one central display to conserve valuable panel space and provide new marketable features. The advances in semiconductor technology have made this concept both technically feasible and cost effective.

## System Description

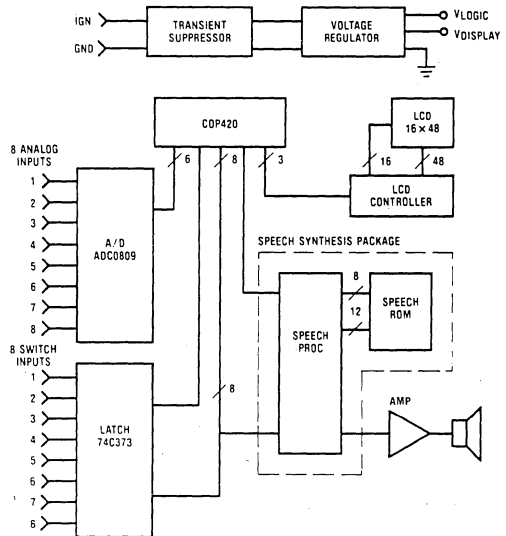
The Diagnostic Display consists of a microcomputer, analog input section, digital input section, liquid crystal display and controller, a speech synthesis package, and a power supply which is outlined on the block diagram. The input section of eight analog channels and eight switch channels was chosen only to demonstrate capability, as the number and mix of analog and digital channels would be tailored to the number of diagnostic messages desired.

From the block diagram, it can be seen that the microcomputer communicates to the liquid crystal display controller via a three-wire bus termed Microwire™. This implies that the display and its controller could be remotely mounted in the instrument cluster, steering wheel, overhead console, etc., while the remainder of the circuitry could be mounted elsewhere under the dashboard.

## Microcomputer

The microcomputer is a National Semiconductor COP 420 which functions as the Diagnostic Display's system controller. The COP 420 is a single-chip processor fabricated using N-channel silicon gate technology. The processor contains 1K x 8 of ROM, 64 x 4 of RAM, clock generator, and 23 input-output lines on board.

In this application, the eight bidirectional L lines are used as a general purpose bus to communicate with the analog-to-digital converter, the switch input latch, and the speech synthesis package. The four G lines are used as chip selects for each of the four peripherals. The four D lines and one IN line are used to control the analog-to-digital converter and to address a particular analog channel. Two additional lines, the SK clock output, and SO serial output line are used to communicate to the liquid crystal display controller.



Diagnostics Display

In normal operation, the microcomputer digitizes and stores all eight analog inputs and stores the states of the eight switch inputs in RAM. If any input is not within programmed limits, it displays the appropriate message and selects the proper verbal phrase. When more than one input is activated simultaneously, the one with the higher priority is selected.

#### Analog Input Section

The analog input section consists of National Semiconductor's ADC0809, which is an eight-bit, eight-channel analog-to-digital converter. This CMOS converter is directly compatible with microprocessor control logic.

The purpose of the A/D converter is to interface with new analog sensors such as outside temperature or paralleling existing sensors such as fuel level.

The threshold levels, where the microcomputer displays a given message, is programmable by the application in software. Although eight inputs are shown, any number could be accommodated to suit the system requirements.

Referring to the block diagram, the analog-to-digital converter is controlled by the microcomputer with six control lines. The control lines address the analog channel, start the conversion, signal the microcomputer when conversion is complete, and enable the TRI-STATE™ drivers. All eight analog values are stored in sixteen four-bit memory locations via the eight-bit data bus. Typical conversion time per channel is 100 microseconds with a maximum total unadjusted error of plus or minus one bit. If additional accuracy is needed, a selected part is available with one half bit accuracy.

#### Digital Input Section

The digital input section consists of a 74C373 CMOS TRI-STATE™ octal latch. Upon command from the microcomputer, the 74C373 latches the input data and outputs it over the eight-bit data bus. The purpose for the digital input section is to input data from mechanical switches such as door jamb or turn signals.

#### Liquid Crystal Display and Controller

The liquid crystal display is a medium area dot matrix multiplexed display. The matrix consists of 16 rows by 48 columns. The display is driven by four CMOS driver circuits, each of which is capable of controlling one quadrant of the display or 8 rows by 24 columns.

The display driver consists of a serial input shift register, an 8x24-bit memory, temperature dependent output drivers, and associated clock circuitry. Communication between the driver circuits and the microcomputer is via a three-wire Microwire™ bus in a serial fashion. The data consists of an address of a dot cluster, the data of whether a dot is on or off, and a read/write bit to indicate whether data is being written or read from memory. Once the memory is loaded with the desired pattern, the display is automatically refreshed by the display driver, so no

further action is required by the microcomputer. Each driver chip also has an input for temperature compensation of the liquid crystal's threshold voltage. The compensation is in the form of a simple variable voltage from a thermistor or similar transducer.

#### Speech Synthesis Package

The speech synthesis package is a system consisting of multiple N-channel devices. It contains a speech processor and speech ROM, and when used with an external filter and amplifier, generates high quality speech.

The speech processor accepts an eight-bit word which is the starting address of the word or phrase to be spoken. Additionally, there is a chip select, write, and interrupt pin to make the part Microbus™ compatible with many microprocessors. An interrupt is generated at the end of any speech sequence, so several sequences or words can be cascaded for additional flexibility.

The speech ROM or ROMs can be as large as 128K bits to be addressed directly by the speech processor. The ROMs can be either static or dynamic clocked types, as the speech processor has a ROM enable pin for use with dynamic ROMs. The ROMs in the package contain the compressed speech data as well as the frequency and amplitude data required for speech output.

#### Power Supply

The power supply in an automotive electronic system is perhaps the most critical part for reliable operation. Its function is to transform the noisy vehicle power to the various voltages required by the system. In the Diagnostics Display, the speech processor requires seven volts, the liquid crystal display requires ten volts, while the rest of the circuit operates at five volts.

In addition to supplying the correct voltages, the power supply must protect the circuit from over-voltages and transients. The LM2930 is the first part in a family of voltage regulators designed for automotive applications. This regulator exhibits a low voltage in to voltage out ratio which provides a constant five volts out, for input voltages as low as 5.6 volts. Additionally, this regulator can accept input voltages to 40 volts, which provides protection against two-battery emergency starts. The large maximum input voltage of 40 volts also simplifies the transient protection network, as now the network needs only to protect the regulator from transients greater than 40 volts.

#### Conclusion

The purpose of the Diagnostics Display is to show a broad design base and present some novel applications for advanced products such as speech synthesis and multiplexed liquid crystal displays. It also shows a 4-bit COP 420 replacing a more costly 8-bit type processor in this application. This is only one example of the many applications of electronics to automotive instrument panels.

# An Electronic Speedometer and Odometer with Permanent Mileage Accumulation

National Semiconductor  
COP Brief 13  
May 1980



## Introduction

As today's automobile becomes more electronic with the addition of engine control systems and digital instrumentation, a need has developed for a method of implementing an electronic odometer that will retain total mileage accumulation information under all conditions, including the loss of vehicle electrical power. This need is made greater by the reduction in available instrument panel space due to downsizing and by a proposed Federal Motor Vehicle Safety Standard requiring tamper-proof odometers.

The requirement of non-volatile mileage storage has been an obstacle for automotive electronic odometer designs. Although an EEPROM (Electrically Alterable Read Only Memory) can be used, they are relatively expensive and have a limited number of erase-write cycles. The system described here uses a fusible link bipolar PROM as the mileage storage device and a low-cost, 4-bit microcontroller as the programming device.

## System Description

A block diagram of the electronic speedometer/odometer is shown in Figure 1. The counting of mileage pulses and the PROM programming are done by a COP 420L, a 4-bit, 1-chip microcontroller (see Figure 2). The mileage pulses are input to the controller through its serial data port. These pulses are counted and stored in RAM. These pulses can be from any type of sensor as long as they have TTL compatible levels.

When the number of pulses counted equals one-tenth of a mile traveled the mileage stored in RAM is updated. The number of pulses equivalent to 0.1 mile is of course dependent on the mileage sensor. The

algorithm for converting from pulses to miles is a software routine and can be modified accordingly to work with various mileage sensors.

A separate count of pulses is kept in another location in RAM for a trip odometer. This mileage can be output on the odometer display by alternate operation of a pushbutton. Another pushbutton clears the trip odometer register.

The speedometer operation is similar to the odometer routine but the updating is dependent on time instead of mileage. The number of pulses counted during a period of time translates to the vehicle speed. A software algorithm converts the number of pulses to speed using a conversion factor dependent on the mileage sensor and display mode selected.

The bipolar PROM is programmed with mileage information when the running mileage count in RAM reaches a predetermined number. The mileage increment that is permanently stored in the PROM is controlled by the operating software and determines the size of the PROM that is required. This is described in more detail in a later section.

When a mileage bit is to be programmed in the PROM, the address of this bit is latched into the address latch by the controller. The proper data for this bit is then put on the 8-bit bus and the proper programming sequence is initiated.

Since the mileage information in the PROM is non-volatile, all operating power is turned off to the circuit when the vehicle ignition is off except for a standby voltage to maintain the trip mileage and running mileage counts stored in the RAM of the controller.

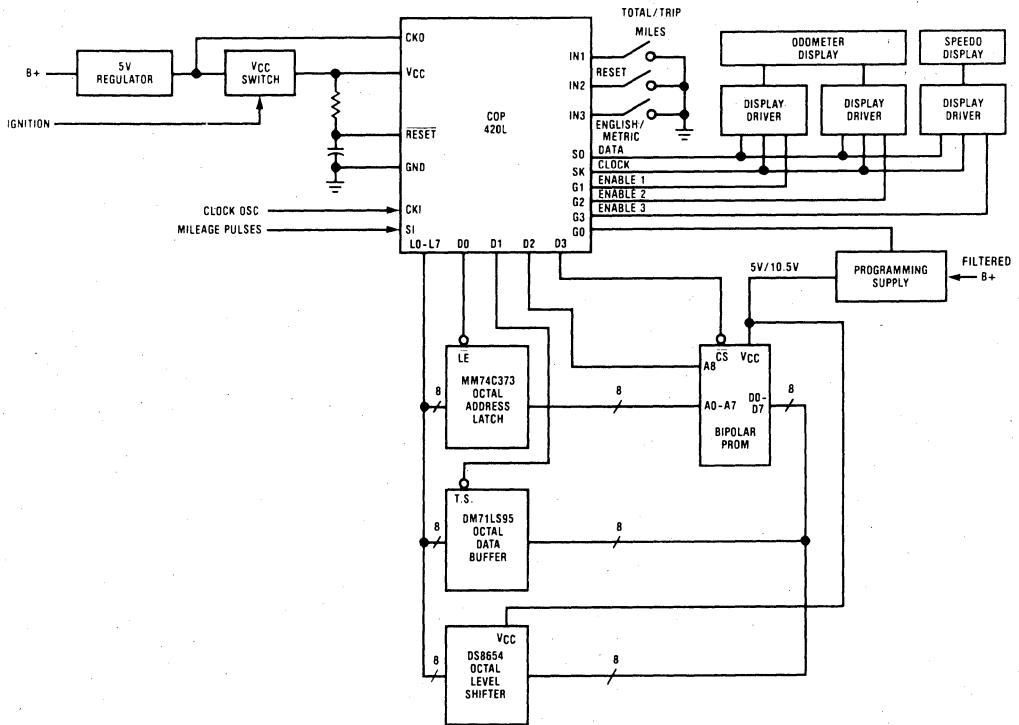


Figure 1. Electronic Speedometer/Odometer

### System Software

Using a microcontroller in an odometer design allows great flexibility of operation and features. The flow chart in Figure 3 is for the prototype speedometer/odometer shown in the block diagram.

When the ignition is turned on, all registers are cleared by the on-chip reset circuitry. After some initial housekeeping, the controller reads a code number from the PROM. This code number is used to provide traceability of the odometer to the vehicle and confirms to the vehicle owner the authenticity of the odometer. The number recorded in the PROM could simply be the vehicle identification number or some other number that has some corresponding vehicle significance. This code number prevents an ingenious individual from replacing the mileage PROM with one of lesser mileage. The number is coded in some manner to prevent easy deciphering.

After this number is displayed for an adequate time, the running mileage in RAM is compared to the total mileage recorded in the PROM. If they are within the predetermined permanent mileage increment the running mileage is accurate and is displayed. If they are not, the RAM has lost data due to a loss of standby power and is restored by transferring the total accumulated mileage recorded in the PROM to the register in RAM. The running mileage is then displayed by the odometer.

The three keys controlling the display mode are read next. Either trip mileage or running mileage is displayed according to the operation of the display key. The trip odometer is cleared when a key depression is detected on the reset button. If a closure is detected on the English/Metric key, a flag is set and all information is displayed in English or Metric units depending on the previous display mode. Next the mileage pulse from the sensor is read from the serial input register. The COP420L has a feature under software control that makes the serial I/O register a binary counter.

In this mode of operation the counter counts high to low level transitions at the SI input. The controller then reads the contents of the register at a rate equal to or greater than the pulse output frequency of the mileage sensor at the maximum vehicle speed. All of the count registers are then incremented.

The mileage registers are examined next. When the pulses counted are equal to 0.1 mile traveled, the trip odometer register and the running mileage register are incremented.

In similar fashion, when the running mileage has accumulated additional mileage equal to the permanent storage increment, the data is programmed into the PROM. The odometer display is updated after the display flags are examined. Either the total

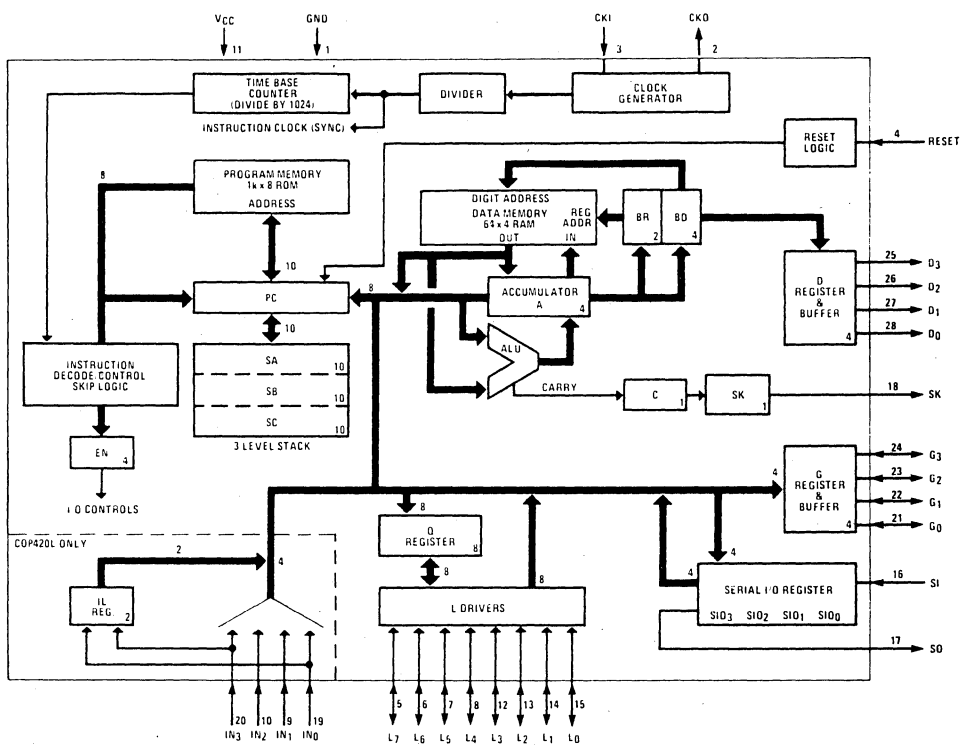


Figure 2. COP 420L/421L Block Diagram

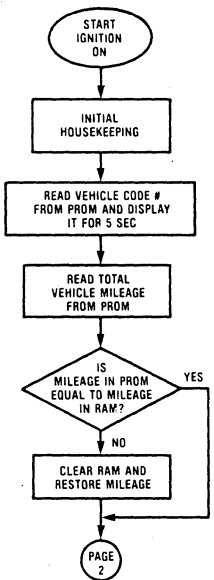


Figure 3. Electronic Speedometer/Odometer Flow Chart

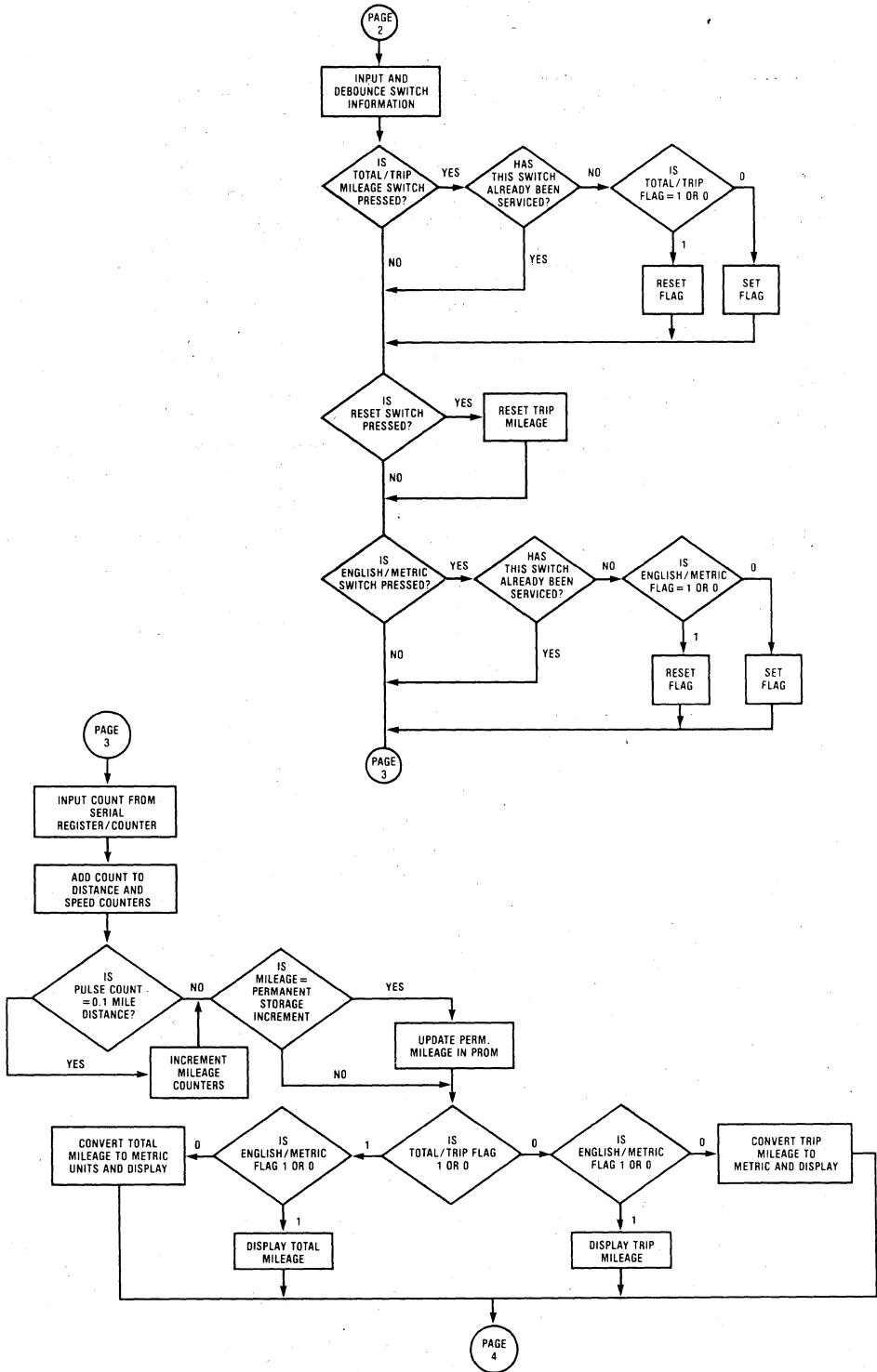


Figure 3. Electronic Speedometer/Odometer (continued)

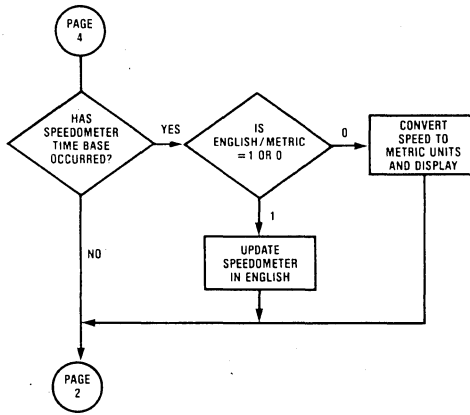


Figure 3. Electronic Speedometer/Odometer (continued)

mileage or trip mileage is displayed in English or Metric units according to the corresponding flag condition.

If the time since the last update of the speedometer is equal to the time base for calculation, the speedometer is updated according to the number of pulses counted during this period. Otherwise, the speedometer reading is not changed.

After this step, the programming returns to reading the display mode switches and continues the loop.

### PROM Selection and Programming

The size of the PROM selected for permanent mileage storage depends on the mileage resolution desired. A 512x8-bit PROM as shown in the block diagram will allow a bit to be programmed every 25 miles for a storage capability of more than 100,000 miles. If 100-mile resolution is adequate, then a 1024-bit PROM could be used, resulting in a lower system cost.

The proper algorithm for programming fusible link PROMs is dependent on the manufacturer and fuse type. However, all types require a voltage for programming that is different from the operating  $V_{CC}$ . This voltage can be provided by the circuit shown in Figure 4.

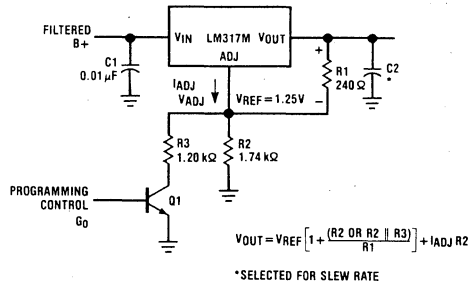


Figure 4. PROM Programming Voltage Regulator





The LM317M regulates by maintaining a reference voltage of 1.25V across R1. Therefore, by changing the voltage at the ADJ pin the regulated output voltage can be varied. During normal operating conditions the output voltage is set to 5.0 volts. Q1 is held on by output G0 of the controller and makes  $V_{ADJ} = 3.75V$ . (Refer to equation in Figure 4.) When the output voltage is to be increased to the required programming voltage, Q1 is turned off and  $V_{ADJ}$  increases to 9.25V. The output then increases to 10.5V, the proper programming voltage for National Semiconductor's bipolar Schottky PROMs. The value of C2 is selected to obtain the proper slew rate of the programming voltage transitions.

When a bit is to be programmed, its address is latched into the MM74C373. The PROM is then disabled and the data for the bit is put on the bus. This data word has a "1" in the proper location for the bit to be programmed and "0s" in the other locations. This "1" turns on the driver in the DS8654 for the respective bit. The programming voltage is then applied by making the G0 output of the COP 420L high. This makes  $V_{CC}$  and the proper output 10.5 volts. The PROM enable line is then taken low for one instruction cycle time (approx. 16 $\mu$ s). Then the voltages are restored to normal operating levels and the bit can be verified by enabling the octal buffer after resetting the L lines. If the bit was not programmed, the programming sequence is repeated until the bit is programmed or it is determined that it will not program and is skipped over.

#### Speedometer and Odometer Displays

The microcontroller interfaces with the speedometer and odometer displays using National Semiconduc-

tor's Microwire™ serial data bus. All display data is sent to the display drivers via the data, clock, and enable lines. This technique allows maximum use of the I/O lines of the microcontroller and also gives great flexibility in choosing the type of display to be used. Table 1 shows a list of National's display drivers that interface by Microwire™.

Table 1.

Device	Package Size	Type of Driver
COP 470	18-pin	4-digit x 8-segment MUX VF
COP 472	20-pin	3 backplane x 12-segment triplexed LCD
*MM54XX	40-pin	32-segment direct drive VF
MM5450	40-pin	35-segment direct drive LED
*MM54XX	40-pin	32-segment direct drive LCD

\*Future product.

#### Summary

By using a low-cost one-chip microcontroller and bipolar PROM, an automotive electronic odometer can be designed with unique features offering permanent, non-volatile mileage accumulation and protection against tampering.

# COP420C Voltage, Current, and Frequency

National Semiconductor  
COP Brief 14  
August 1981



The following curves are presented in order to show the relationship of the voltage, current, and frequency on the COP420C chip. Included are six curves and one diagram.

Curves 1, 2, and 3 give the maximum current versus voltage at different frequencies. They are given for the divide by 8, 16, and 32 modes. Note that these curves are not valid if the R/C oscillator option is selected. Figure 1 shows the setup used to measure the current in curves 1, 2, and 3.

Curve 4 gives the maximum current versus voltage when the COP420C is in the idle state.

Curve 5 shows the maximum operating frequency versus voltage of the COP420C and the COP320C.

Curve 6 shows the *typical* current drain of the COP420C when running off an R/C oscillator.

## CKI Oscillator Input

The signal present at the CKI input has a large effect on the power drain of the COP420C. In curves 1 to 4, CKI is a square wave clock that swings rail to rail. If, for example, CKI is a sine wave input, the COP420C will draw additional current. The following chart shows the amount of extra current that is *typically* drawn with a sine wave clock on CKI input.

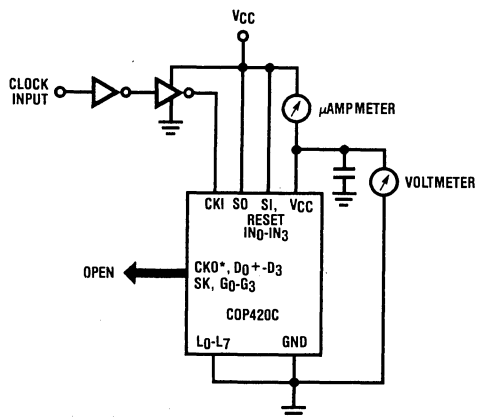
Volts	Extra Current
6	175 $\mu$ A
5	100 $\mu$ A
4	50 $\mu$ A
3	25 $\mu$ A
2.4	10 $\mu$ A

System Current Drain

## System Current Drain

The current drain of the COP420C in an operating system may be more than the values shown on the curves. This can be caused by the following:

1. Any input which is not within 0.3V of ground or  $V_{CC}$  will draw some current. For example, if  $CKI = 1.9V$  at  $V_{CC} = 5V$ , the COP420C can draw an extra 1 milliamp! Other inputs will be about  $\frac{1}{3}$  less, but will still add an appreciable amount of current drain if the inputs are at half levels.
2. A floating input can drift to a half level and draw extra current. No inputs should be floating on a CMOS part.
3. Any input with an internal load device will source current if not at  $V_{CC}$  level.
4. A slow rise or fall time on an input will draw current because the input will be at the half level for some period of time.
5. An output sourcing current will do so from the  $V_{CC}$  supply.
6. An output switching a capacitive load at a fast frequency will increase the current drain (AC power).

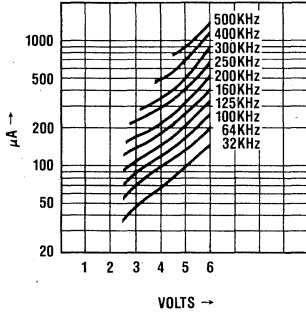


\*Connect CKO to  $V_{CC}$  if an input

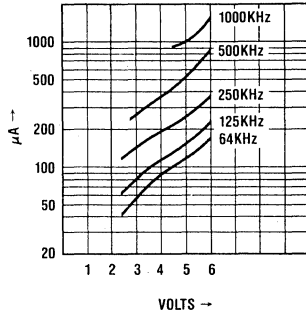
\*\*Ground D<sub>0</sub> if Dual Clock

Figure 1. Connection Diagram to Measure COP420C Current

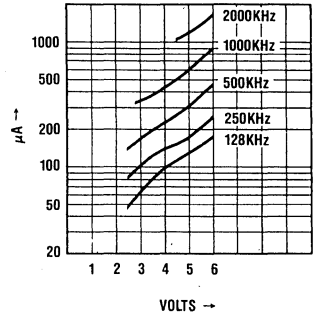
Maximum Current vs Voltage (÷ 8 Mode)



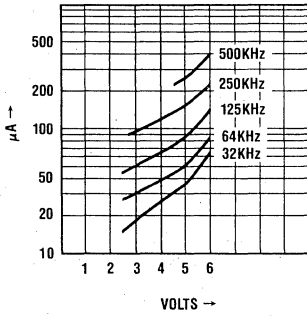
Maximum Current vs Voltage (÷ 16 Mode)



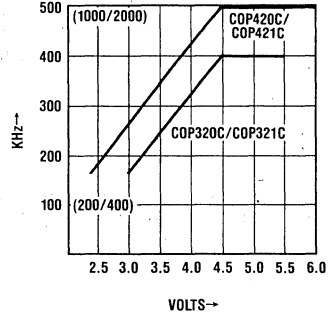
Maximum Current vs Voltage (÷ 32 Mode)



Maximum Current in Idle State vs Voltage (÷ 8 Mode)



Maximum Operating Frequency vs Voltage





Section 10  
**Appendix/  
Physical Dimensions**

**10**



# COP420-HGZ/N Preprogrammed Single-Chip Microcontroller for Musical Organ

## Features and Functions

**Play Mode:** Twenty-five musical keys and 25 LEDs are provided to denote F to F with half notes in between. All the keys and LEDs are directly detected and driven by the microprocessor. Depression of the key will give the corresponding musical note and light up the corresponding LED.

**Clear:** Memory is provided to store a played tune. Depression of the CLEAR key erases the memory and the microprocessor is ready to store new musical notes. A maximum of 28 notes can be stored where each note can be of one to eight musical beats. (Two bytes of memory are required to store one musical note. Any note longer than eight musical beats will require additional memory space for storage.)

**Playback:** Depression of this button will playback the tune stored in the memory since last "clear."

**Preprogrammed Tunes:** There are ten preprogrammed tunes (each has an average of 55 notes) masked in the chip. Any tune can be recalled by depressing the "Tune Button" followed by the corresponding "Sharp-Key."

**Learn Mode:** This mode is for the player to learn the ten preprogrammed tunes. By pressing the "Learn Button"

followed by the corresponding "Sharp Key," the LEDs will be lighted up one by one to indicate the notes of the selected tune. The LED will remain "on" until the player presses the correct musical key; the LED for the next note will then be lighted up.

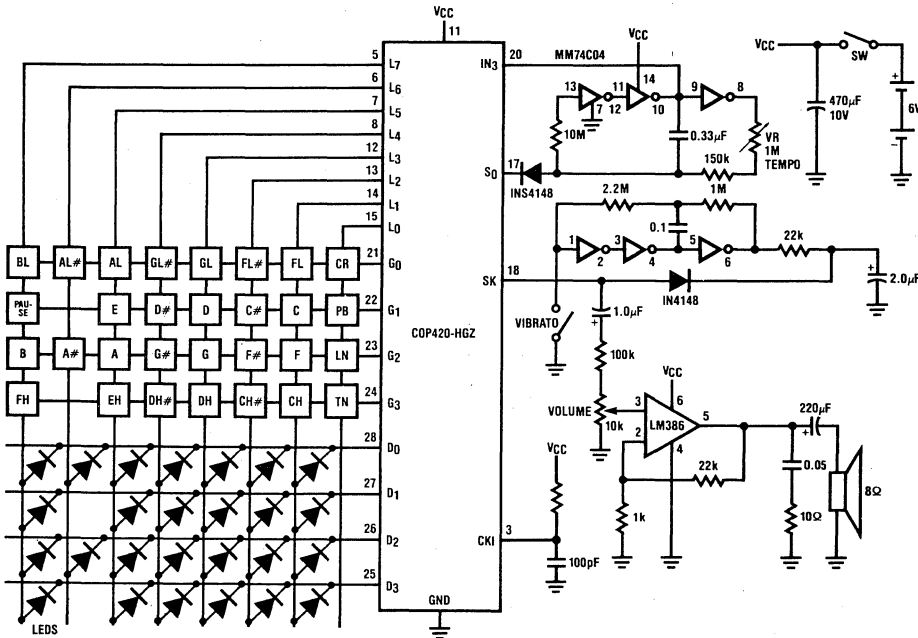
**Pause:** In addition to the 25 musical keys, there is a special pause key. The depression of this key generates a blank note to the memory.

**Note:** In the Learn Mode when playing "Oh Susanna," the pause key must be used.

**Tempo:** This is a control input to the musical beat time oscillator for varying the speed of the musical tunes.

**Vibrato:** This is a switch control to vary the frequency vibration of the note.

**Tunes Listing:** The following is a listing of the ten preprogrammed tunes: 1) Jingle Bells, 2) Twinkle, Twinkle Little Star, 3) Happy Birthday, 4) Yankee Doodle, 5) Silent Night, 6) This Old Man, 7) London Bridge Is Falling Down, 8) Auld Lang Syne, 9) Oh Susanna, 10) Clementine.



Circuit Diagram of COP420 Musical Organ



# COP420L-HSB Digital Tuning System Controller

## General Description

The COP420L-HSB is an N-channel microcontroller dedicated to digital tuning systems. It is fabricated using N-channel, silicon-gate MOS technology. The controller provides all system timing, logic and I/O necessary to interface with DS8906N and other MICROWIRE™ compatible devices in a digital tuning system. Features include single supply operation, ultra slow keyboard matrix frequency and options catered for different applications. Standard test procedures and reliable high-density fabrication techniques provide manufacturers a features-packed system at a low end cost.

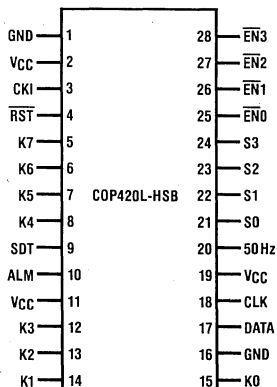
A digital tuning system provides the following which could only be realized with expensive and complicated circuitry before.

- Precise tuning of station frequency
- Digital display of exact frequency
- Electronic storage of multiple stations in memory
- Immediate access of preferred frequencies
- Automatic station searching
- Full function clock

## Features

- Low cost
- Dual speed manual Up/Down tuning
- Automatic memory scanning
- Automatic Up/Down station searching
- Ten band independent memories
- Power up last station recall
- Strap selectable for USA or European band
- 5-digit resolution for FM band
- Memory Store mode indicator
- 12/24-hour crystal controlled clock
- 59-minute sleep timer
- 24-hour auto turn off alarm
- Multiplex or direct drive displays
- Single or Dual display modes
- MICROWIRE™ compatible
- 4 x 8 interference free matrix keyboard

MICROWIRE is a trademark of National Semiconductor Corp.



DUAL-IN-LINE PACKAGE

### Top View

Order No. COP420L-HSBN  
NS Package Number N28A

Figure 1. Connection Diagram

Pin	Description
K0-K7	Matrix keyboard input
S0-S3	Keyboard scan output
EN0-EN3	Chip enable for slave devices
CKI	External clock input
RST	Reset
SDT	Station Detect input
ALM	Alarm On/Off
CLK	MICROWIRE™ clock
DATA	MICROWIRE data
50Hz	External time base input



## Absolute Maximum Ratings

Voltage at Any Pin Relative to GND	-0.5V to +10V
Ambient Operating Temperature	0°C to +70°C
Ambient Storage Temperature	-65°C to +150°C
Lead Temperature (Soldering, 10 seconds)	300°C
Power Dissipation	0.75 Watt at 25°C 0.4 Watt at 70°C

*Absolute maximum ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications are not ensured when operating the device at absolute maximum ratings.*

## DC Electrical Characteristics

$0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 6.3\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Operating Voltage ( $V_{CC}$ )	Note 1	4.5	6.3	V
Operating Supply Current	(All inputs and outputs open)		8	mA
Power Supply Ripple	peak to peak		0.5	V
Input Voltage Levels				
CKI Input Levels				
Logic High ( $V_{IH}$ )		2.0		V
Logic Low ( $V_{IL}$ )		-0.3	0.4	V
RESET Input Levels				
Logic High		0.7 $V_{CC}$		V
Logic Low		-0.3	0.6	V
K Inputs				
Logic High		3.6		V
Logic Low		-0.3	1.2	V
All Other Inputs				
Logic High	$V_{CC} = 5\text{V} \pm 10\%$	2.0		V
Logic Low		-0.3	0.8	V
Input Capacitance			7.0	pF
Hi-Z Input Leakage		-1.0	+1.0	$\mu\text{A}$
Output Voltage Levels	$V_{CC} = 5\text{V} \pm 5\%$			
Logic High ( $V_{OH}$ )	$I_{OH} = -25\mu\text{A}$	2.7		V
Logic Low ( $V_{OL}$ )	$I_{OL} = 0.36\text{mA}$		0.4	V
Output Current Levels				
Output Sink Current				
CLK and DATA Outputs ( $I_{OL}$ )	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.9		mA
EN Outputs ( $I_{OL}$ )	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 0.4\text{V}$	0.4		mA
S Outputs ( $I_{CL}$ )	$V_{CC} = 4.5\text{V}$ , $V_{OL} = 1.0\text{V}$	15		mA
Output Source Current				
Standard Configuration				
K Inputs ( $I_{OH}$ )	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 2.0\text{V}$	-30	-250	$\mu\text{A}$
CLK and DATA Outputs	$V_{CC} = 4.5\text{V}$ , $V_{OH} = 1.0\text{V}$	-1.2		mA

Note 1:  $V_{CC}$  voltage change must be less than 0.5V in a 1ms period to maintain proper operation.

## AC Electrical Characteristics

$0^{\circ}\text{C} \leq T_A \leq +70^{\circ}\text{C}$ ,  $4.5\text{V} \leq V_{CC} \leq 6.3\text{V}$  unless otherwise noted.

Parameter	Conditions	Min.	Max.	Units
Instruction Cycle Time ( $t_C$ )		16		$\mu\text{s}$
CKI				
Input Frequency ( $f_i$ )	- 8 mode		0.5	MHz
Duty Cycle		30	60	%
Rise Time			120	ns
Fall Time			80	ns
Keybounce Time		32		ms

tion and the other showing frequency information. Whereas in conventional single display systems, the display shows both time and frequency information in a time-sharing method. The National system provides a time-prioritized display-sharing method. That is, whenever a tuning function is completed, the frequency information will stay on the display for eight seconds then time display will take over. This is achieved by using EN3 for the driver's enable logic.

**Control Outputs**

Six open collector outputs controlled by the COP420L-HSB are provided from DS8906N, the phase lock loop for controlling radio switching circuits.

**Radio ON/OFF:** A high from this output indicates that the radio should be switched on and vice versa.

**AM/FM:** Output for controlling the AM/FM bandswitch. A high level output indicates FM and a low indicates the AM band.

**MUTE:** For muting the audio output when performing any frequency related function. The output will go high prior to the frequency change except when doing fine tuning.

**ALARM ENABLE:** Active high output for turning on the alarm circuit at alarm time.

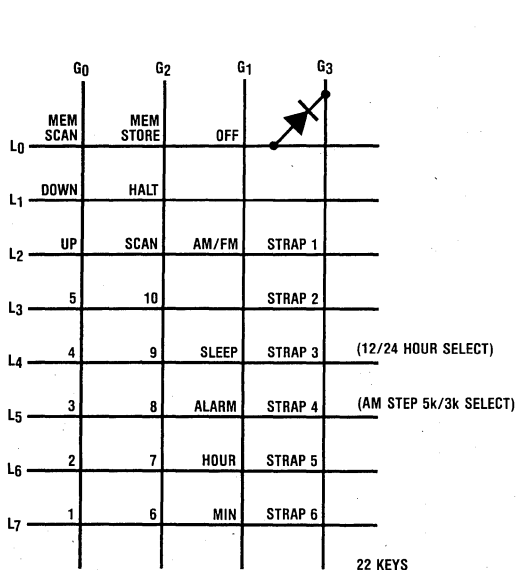
**50kHz IND:** For driving the 50kHz indicator in FM band or the LSB in a 5-digit display. Output is active high.

**MEM STORE IND:** For driving the memory store mode indicator. Output is active high.

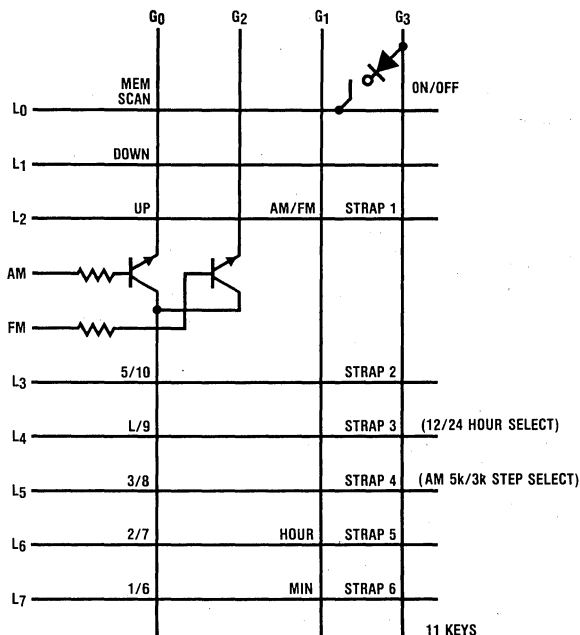
**Typical Implementation Alternatives**

A full keyboard or any portion of it can be implemented with various applications for features/functions vs. cost/size.

Figure 11 shows two keyboard configurations with 22-key and 11-key keyboards for a desk top/tuner system or auto-radio system respectively.



Desk Top DTR Keyboard



Car DTR Keyboard

Figure 11.

## Functional Description

### Logic I/Os

**CKI Input:** This input accepts an external 500kHz signal, divides it by eight and outputs the quotient at the CLK output as the system clock.

**RST Input:** Schmitt trigger input to clear device upon initialization.

**SDT Input:** Interrupt input for station detection. The SDT signal is generated by the radio's station detector and used by the COP420L-HSB to determine if there is a valid station on the active frequency. The status of the SDT input is only relevant during station searching mode. A high on SDT will temporarily terminate the search mode for eight seconds.

**ALM Input:** A high on ALM will activate alarm output via slave device at alarm time. A low on the input will disable alarm function.

**DATA Output:** Push-pull output providing serial data to external devices.

**CLK Output:** Push-pull output providing system clock at data transmitting time.

**50Hz Input:** A normally high input to accept a 50Hz external time base for real-time calculation.

### Momentary Keys Description

**MEM 1-MEM 10:** Each memory represents data of a favorite station in a certain band. Depression of one of these keys will recall the previous stored data and transmit it to the PLL. The PLL will in turn change the radio's receiving frequency as well as the band if necessary. Memory recall keys can also turn on the radio.

**UP:** This key will manually increment receiving frequency. The first four steps of increment will be for fine tuning a station, after which will be fast slewing meant for manual receive frequency changing.

**DOWN:** Has the same function as UP key except that frequency is decremented.

**MEMORY SCAN:** This will start the radio scanning through all ten memories automatically at eight seconds per memory starting from Memory 1. This will also turn on the radio if it was off.

**MEMORY STORE:** Enables the memory store mode which lasts for three seconds. Depression of any memory key will store the active frequency and band in that memory and disable the store mode. Any function key will also disable the mode to prevent memory data being accidentally destroyed.

**HALT:** Depression of the HALT key will stop the search and scan functions at current frequency or memory. HALT also turns on the radio during off time and recall frequency display in single display mode.

**SEARCH:** Activates station searching in the current band. Search speed is 50ms per frequency step with wrapping around at end of band. An 8-second stop will take place on reaching a valid station. The HALT key or any function key will terminate the search. Search direction will normally be upwards unless the DOWN key has been depressed prior to the SEARCH key or during the search function in which case search direction will be downwards.

**OFF:** Turns off the radio or alarm when active.

**AM/FM:** Radio band switch.

**SLEEP:** Activates sleep mode, turns on radio on depression and off radio at the end of sleep period. Setting of sleep period is done by depressing the SLEEP and MINUTE key simultaneously.

**ALARM:** Enables alarm time setting. Depressing the HOUR or MINUTE key and ALARM key simultaneously will set the alarm hour and minute respectively.

**HOUR:** Sets the hour digits of time-related functions.

**MINUTE:** Sets the minute digits of time-related functions.

### Diode Straps Connections

**STRAP 0:** Controls the on and off of radio. In applications where a toggle type ON/OFF switch is used, momentary OFF key can be omitted; connecting the strap will turn on the radio and vice versa. Must be connected to use momentary OFF key.

**STRAP 1, 2:** Selects the AM IF options.

**STRAP 2:** 12/24-hour clock select.

**STRAP 4:** 3/5kHz AM step size select.

**STRAP 5, 6:** FM IF offsets select.

	STRAP 0	STRAP 3	STRAP 4
Connected	Radio ON	12 hour	5kHz step
Open	Radio OFF	24 hour	3kHz step

### AM/FM IF Options:

	AM	STRAP 1	STRAP 2
455kHz		X	X
460kHz		X	✓
450kHz		✓	X
260kHz		✓	✓

	FM	STRAP 5	STRAP 6
10.7MHz		X	X
10.75MHz		X	✓
10.65MHz		✓	X
10.8MHz		✓	✓

X = No connection.

✓ = Diode inserted.

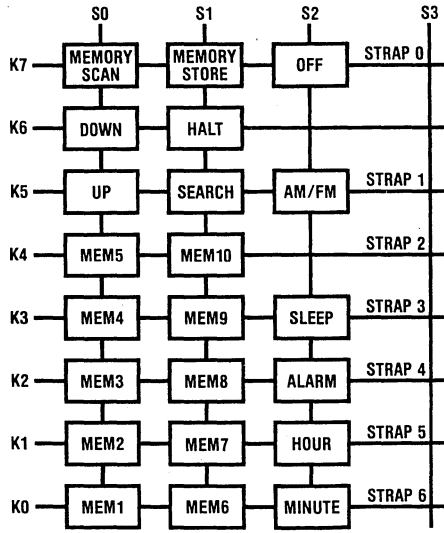
### Indirect Features and Options

As indicated in Figure 10, there are a few options and indirect features provided via the help of a slave device, namely the Phase Lock Loop, DS8906N.

### Display Options

As mentioned above, the COP420L-HSB is MICRO-WIRE® compatible. Internal circuitry enables it to directly interface with all of National's serial input MICRO-WIRE compatible display drivers whether they are of a direct drive or multiplex drive format. On Figure 11 is a list of drivers available for the system. EN1 and EN2 are optional enable outputs meant for a dual display system in which EN3 will not be used. By dual display, it means that one display will be constantly showing time informa-

Digitally Tuned Radio Controller and Clock



Keyboard Matrix Configuration

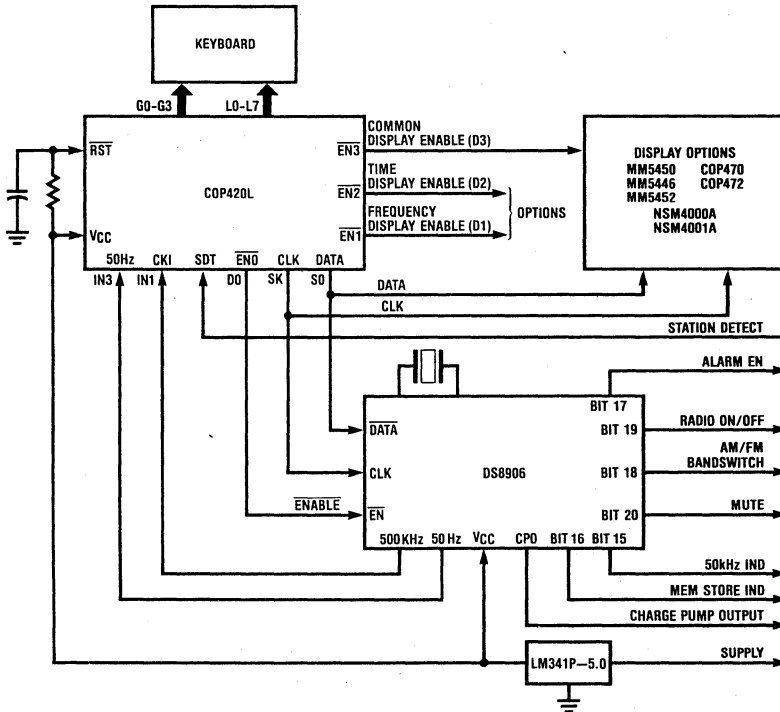
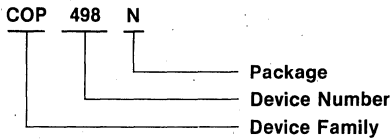
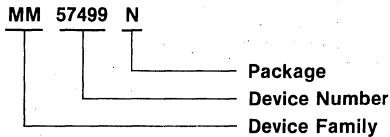
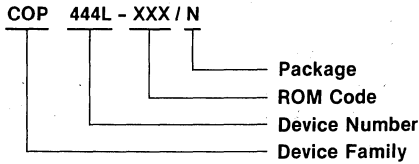


Figure 10. Digital Tuning System Block

# Ordering Information/Physical Dimensions



## Package

- D — Glass/Metal Dual-In-Line Package
- J — Ceramic Dual-In-Line Package
- N — Epoxy Dual-In-Line Package

## ROM Code

- COPS — Magnetic Disk, PROM, or Tape
- Contact your local sales-office for submittal procedures.

## Device Number

- 4-, 5-, or 6-Digit Number Suffix Indicators

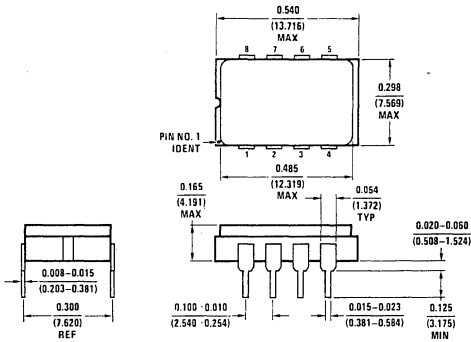
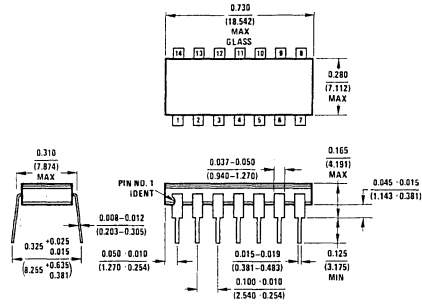
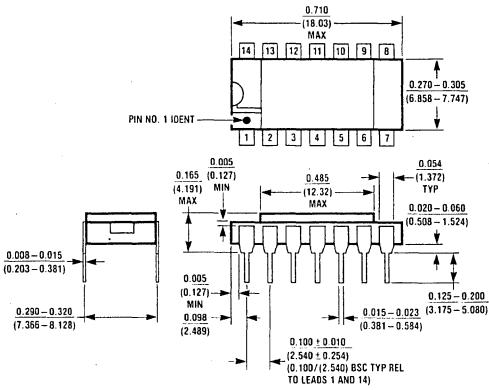
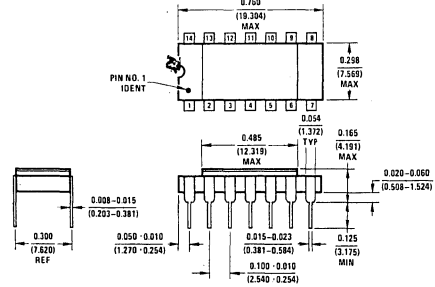
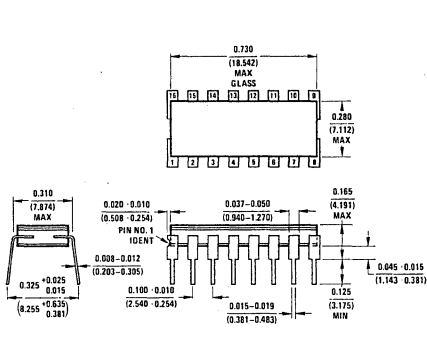
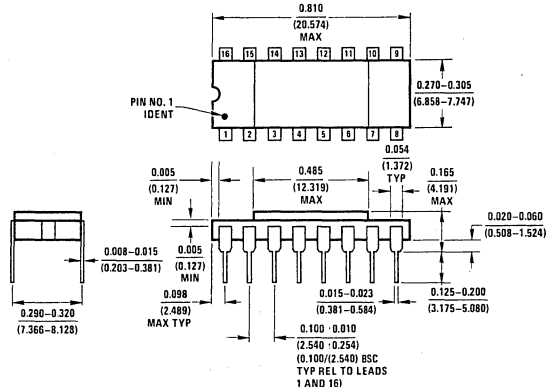
## Device Family

- MM — MOS Monolithic
- COP— Controller Processor

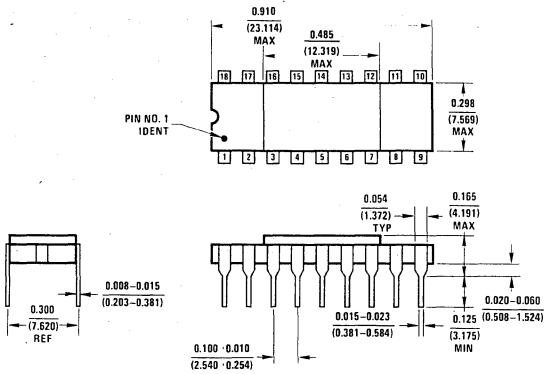
## Packages

### Dual-In-Line Packages

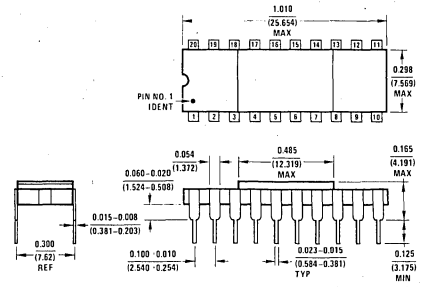
- (N) Devices ordered with "N" suffix are supplied in molded dual-in-line packages. Molding material is EPOXY B, a highly reliable compound suitable for military as well as commercial temperature range applications. Lead material is Alloy 42 with a hot solder dipped surface to allow for ease of solderability.
- (J) Devices ordered with the "J" suffix are supplied in ceramic dual-in-line packages. The body of the package is made of ceramic and hermeticity is accomplished through a high temperature sealing of the package. Lead material is tin-plated kovar.
- (D) Devices ordered with the "D" suffix are supplied with glass/metal dual-in-line packages. The top and bottom of the package are gold-plated kovar, as are the leads. The side walls are glass, through which the leads extend, forming a hermetic seal.


**8-Lead Cavity DIP (D)  
NS Package D08C**

**14-Lead Cavity DIP (D)  
NS Package D14A**

**14-Lead Cavity DIP (D)  
NS Package D14D**

**14-Lead Cavity DIP (D)  
NS Package Number D14E**

**16-Lead Cavity DIP (D)  
NS Package D16A**

**16-Lead Cavity DIP (D)  
NS Package D16C**

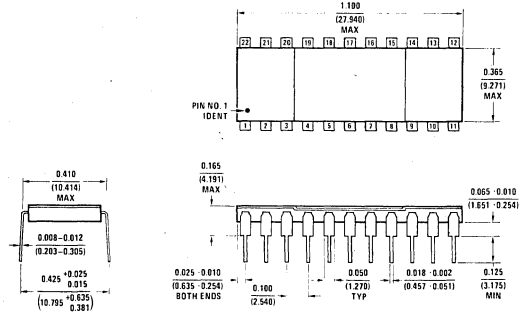
Physical Dimensions



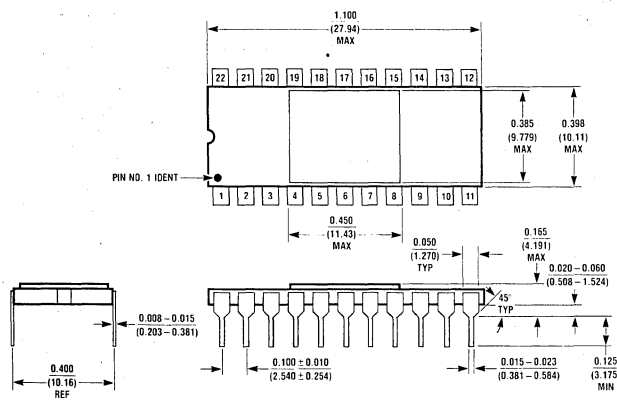
18-Lead Cavity DIP (D)  
NS Package D18A



20-Lead Cavity DIP (D)  
NS Package D20A

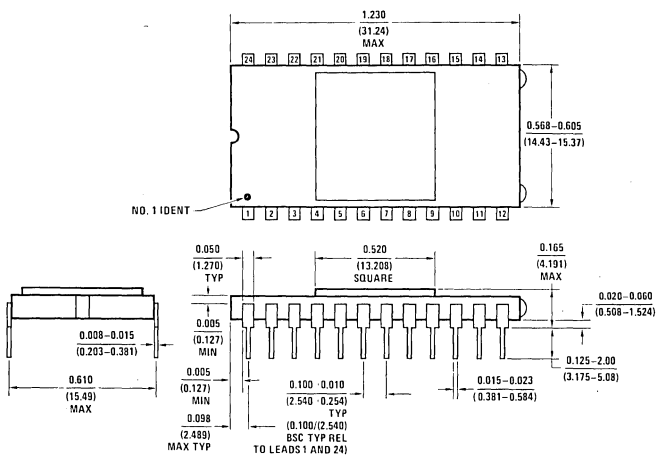


22-Lead Cavity DIP (D)  
NS Package D22A

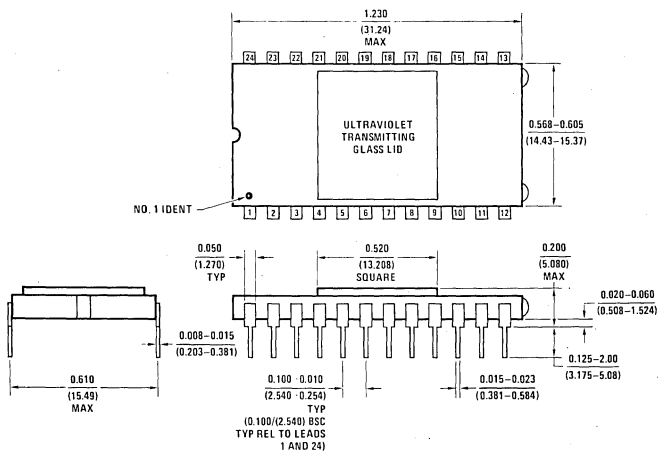


22-Lead Cavity DIP (D)  
NS Package D22C

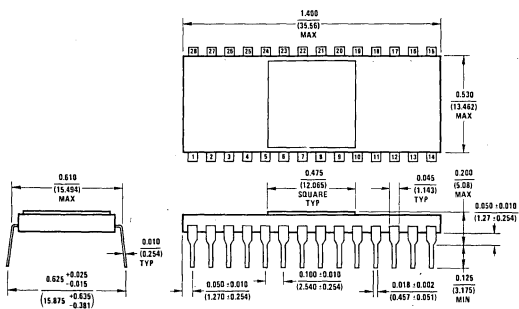
Physical Dimensions



24-Lead Cavity DIP (D)  
NS Package D24C



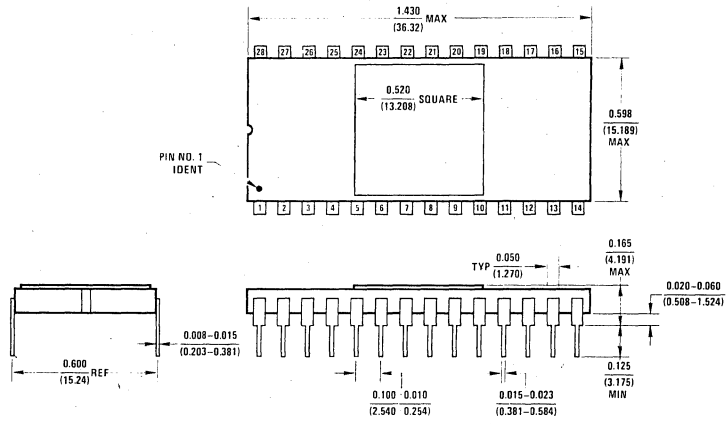
24-Lead Cavity DIP (D)  
NS Package D24CQ



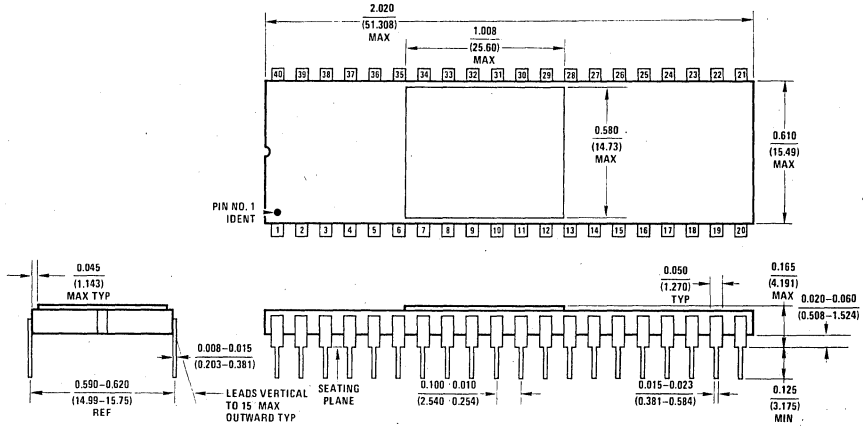
28-Lead Cavity DIP (D)  
NS Package D28A



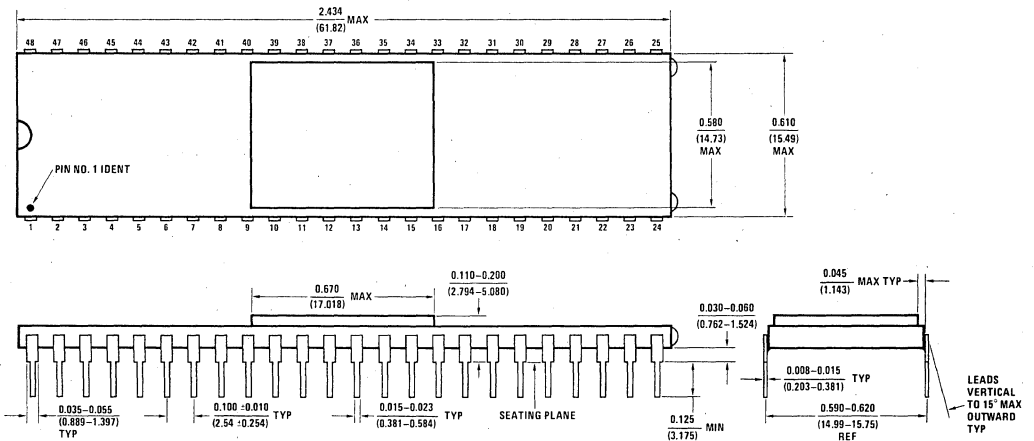
Physical Dimensions



28-Lead Cavity DIP (D)  
NS Package D28C

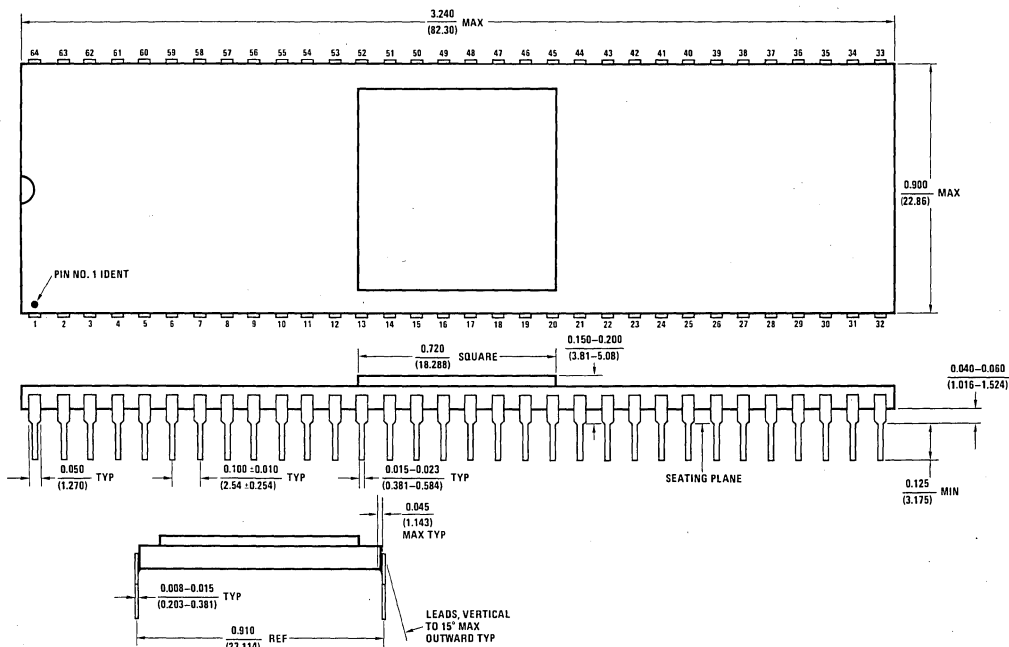


40-Lead Cavity Dip (D)  
NS Package D40C

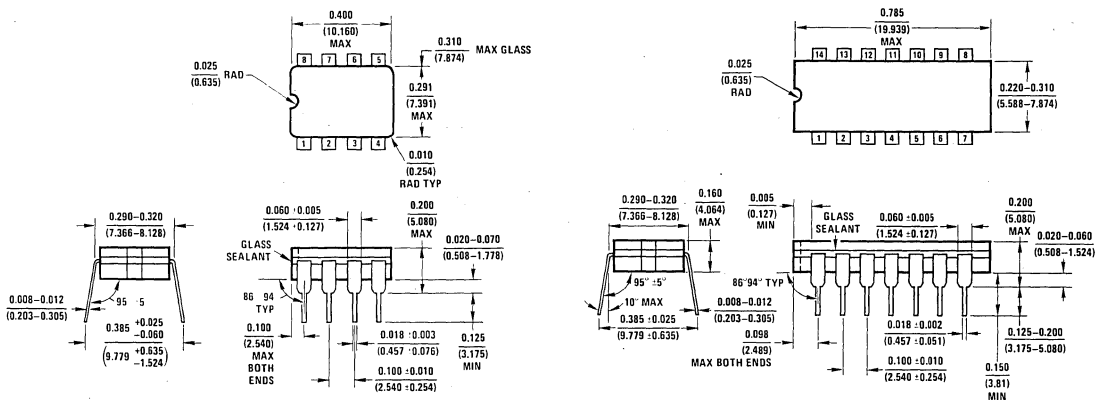


48-Lead Cavity DIP (D)  
NS Package D48A

# Physical Dimensions



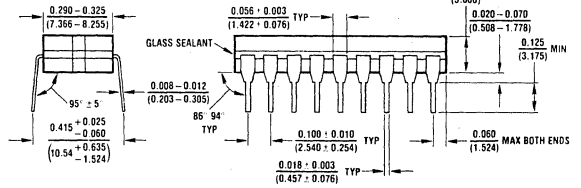
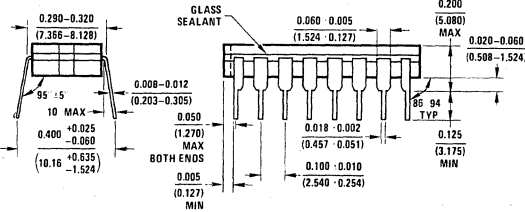
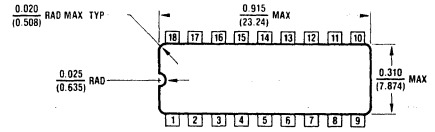
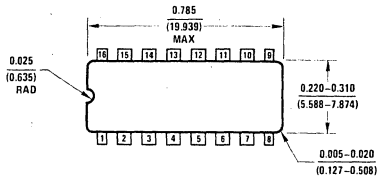
**64-Lead Cavity DIP (D)  
NS Package D64A**



**8-Lead Cavity DIP (J)  
NS Package J08A**

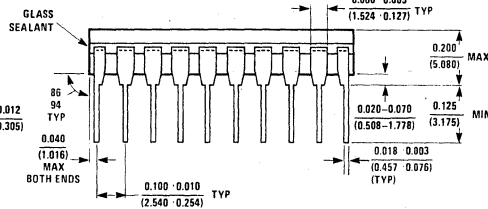
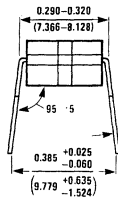
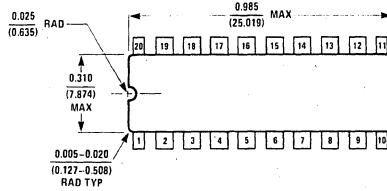
**14-Lead Cavity DIP (J)  
NS Package J14A**

# Physical Dimensions

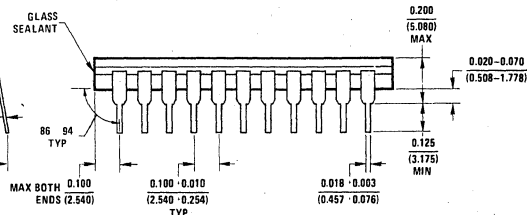
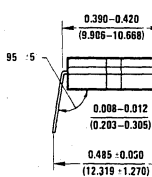
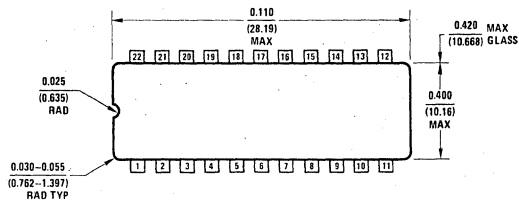


**16-Lead Cavity DIP (J)  
NS Package J16A**

**18-Lead Cavity DIP (J)  
NS Package J18A**



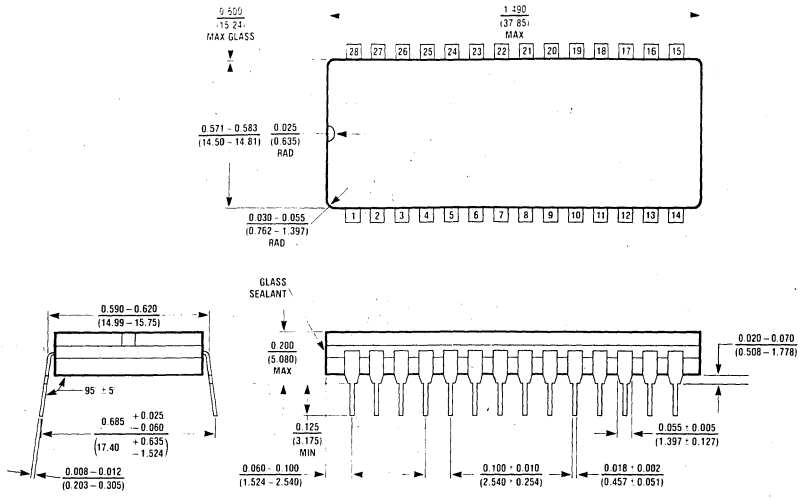
**20-Lead Cavity DIP (J)  
NS Package J20A**



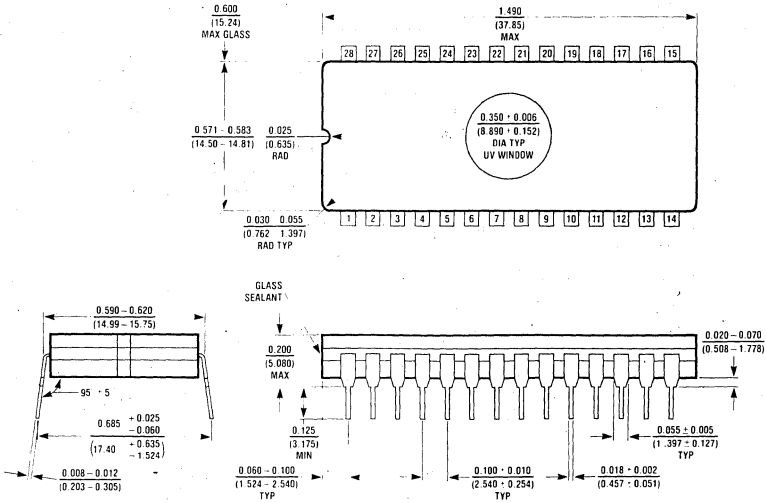
**22-Lead Cavity DIP (J)  
NS Package J22A**



Physical Dimensions



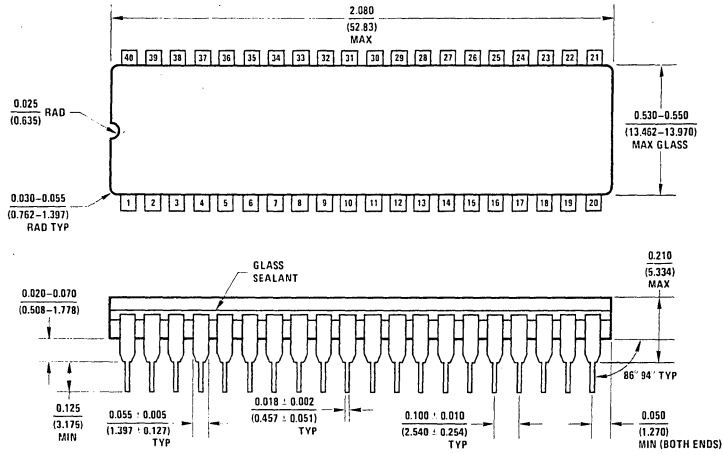
28-Lead Cavity DIP (J)  
NS Package J28B



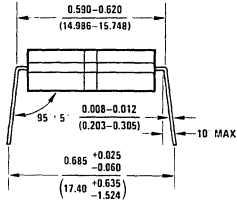
28-Lead Cavity DIP (J)  
NS Package J28BQ

# Physical Dimensions

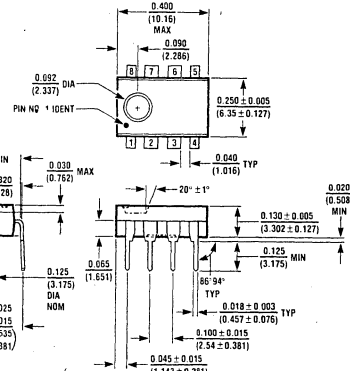
Physical Dimensions



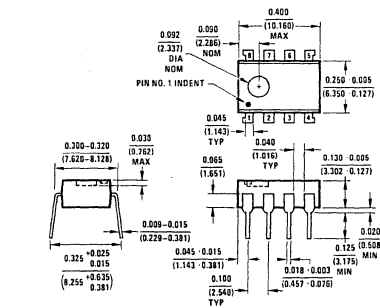
**40-Lead Cavity DIP (J)  
NS Package J40A**



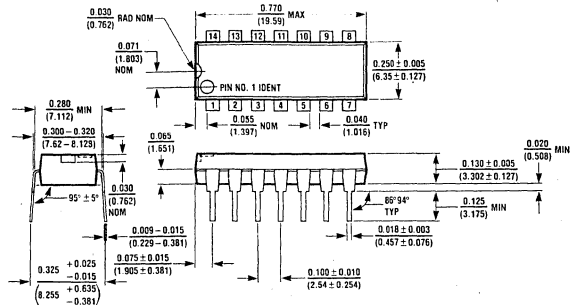
**8-Lead Molded DIP (N)  
NS Package N08A**



**8-Lead Molded DIP (N)  
NS Package N08E**



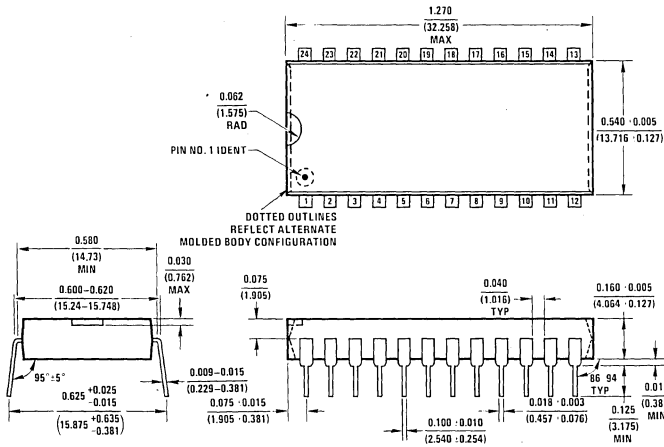
**14-Lead Molded DIP (N)  
NS Package N14A**



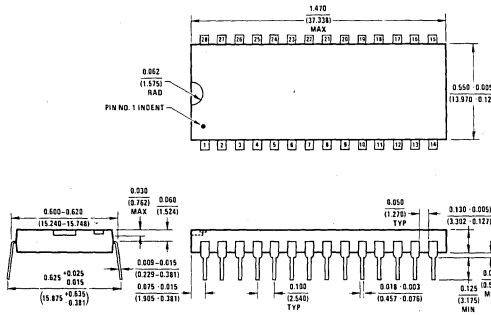
**16-Lead Molded DIP (N)  
NS Package N14E**



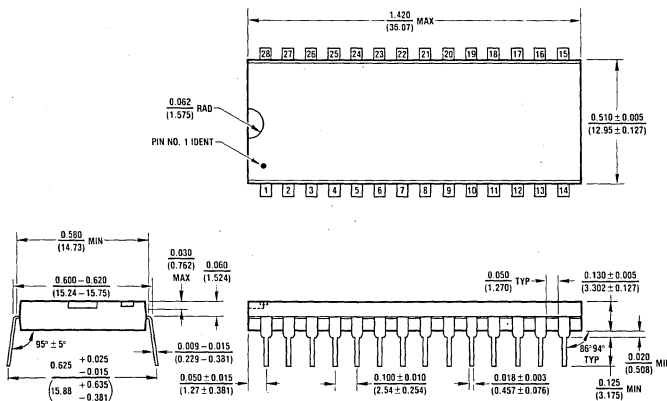
# Physical Dimensions



**24-Lead Molded DIP (N)  
NS Package N24A**



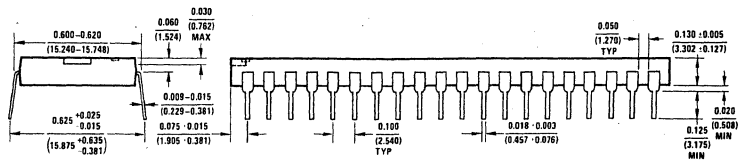
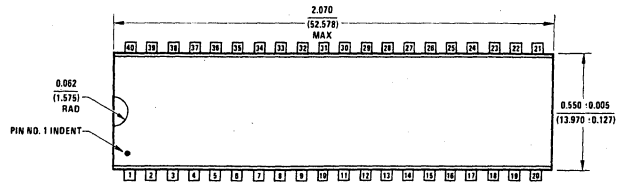
**28-Lead Molded DIP (N)  
NS Package N28A**



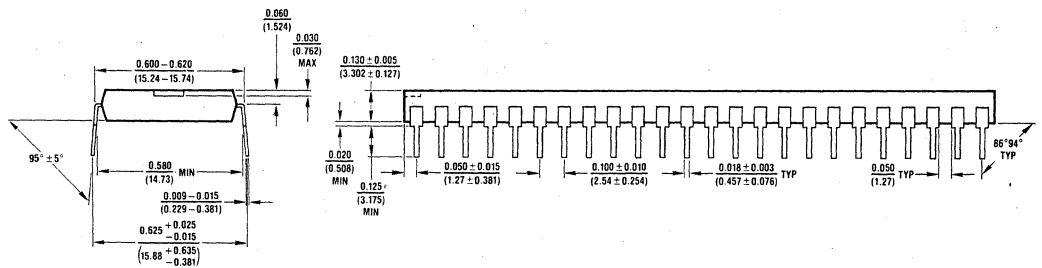
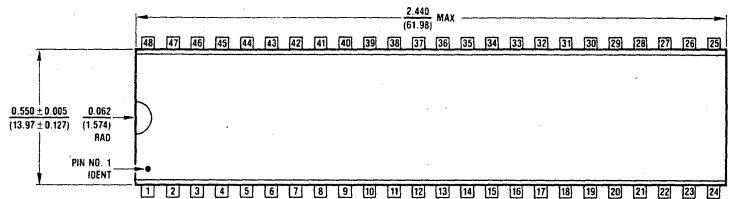
**28-Lead Molded DIP (N)  
NS Package N28B**



Physical Dimensions

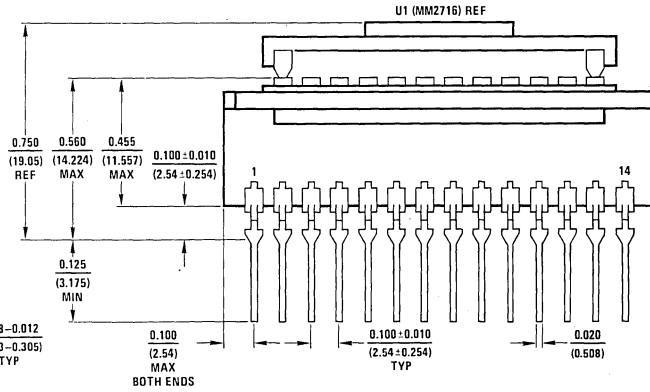
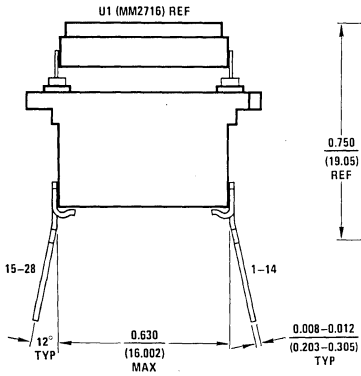
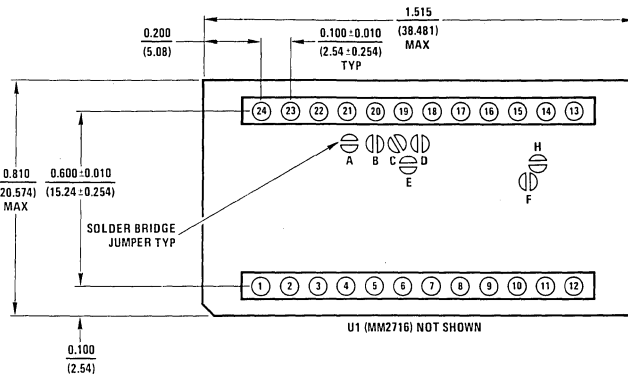


40-Lead Molded DIP (N)  
NS Package N40A

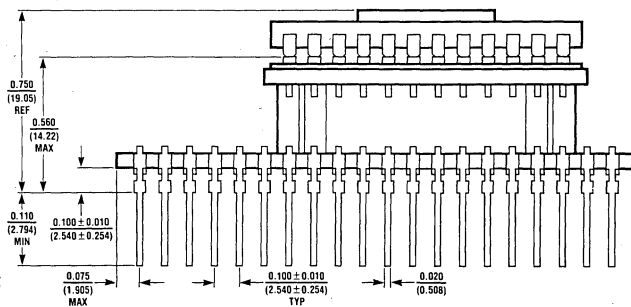
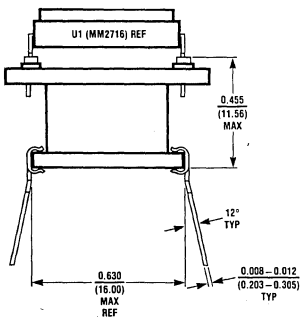
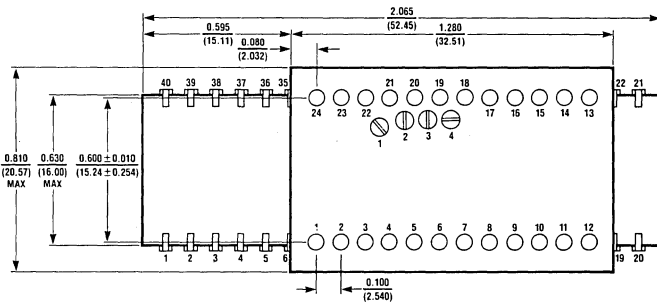


48-Lead Molded DIP (N)  
NS Package N48A

# Physical Dimensions



**Piggyback Package  
COP420R**



**Piggyback Package  
COP440R**

# Notes

**National Semiconductor Corporation**

2900 Semiconductor Drive  
Santa Clara, California 95051  
Tel: (408) 721-5000  
TWX: (910) 339-9240

**Electronica NSC de Mexico SA**

Hegel No. 153-204  
Mexico 5 D.F. Mexico  
Tel: (905) 531-1689, 531-0659  
Telex: 017-73550

**NS Electronics Do Brasil**

Avda Brigadeiro Faria Lima 830  
8 Andar  
01452 Sao Paulo, Brasil  
Telex: 1121008 CABINE SAO PAULO  
113193 INSBR BR

**National Semiconductor GmbH**

Furstenriederstraße Nr. 5  
8 München 21  
West Germany  
Tel.: (089) 56 01 20  
Telex: 522772

**National Semiconductor (UK), Ltd.**

301 Harpur Centre  
Horne Lane  
Bedford MK40 1TR  
United Kingdom  
Tel: 0234-47147  
Telex: 826 209

**National Semiconductor Benelux**

Ave. Charles Quint 545  
1080 Brussels  
Belgium  
Tel: (02) 4661807  
Telex: 61007

**National Semiconductor (UK), Ltd.**

1, Bianco Lunos Allé  
DK-1868 Copenhagen V  
Denmark  
Tel: (01) 213211  
Telex: 15179

**National Semiconductor**

Expansion 10000  
28, Rue de la Redoute  
92 260 Fontenay-aux-Roses  
France  
Tel: (01) 660-8140  
Telex: 250956

**National Semiconductor S.p.A.**

Via Solferino 19  
20121 Milano  
Italy  
Tel: (02) 345-2046/7/8/9  
Telex: 332835

**National Semiconductor AB**

Box 2016  
12702 Skärholmen  
Sweden  
Tel: (08) 970190  
Telex: 10731

**National Semiconductor**

Calle Nunez Morgado 9  
Esc. Dcha. 1-A  
Madrid 16  
Spain  
Tel: (01) 733-2954/733-2958  
Telex: 46133

**National Semiconductor Switzerland**

Alte Winterthurerstrasse 53  
Postfach 567  
CH-8304 Wallisellen-Zürich  
Tel: (01) 830-2727  
Telex: 59000

**National Semiconductor**

Pasilanraatio 6C  
00240 Helsinki 24  
Finland  
Tel: (0) 14 03 44  
Telex: 124854

**NS Japan K.K.**

POB 4152 Shinjuku Center Building  
1-25-1 Nishishinjuku, Shinjuku-ku  
Tokyo 160, Japan  
Tel: (03) 349-0811  
TWX: 232-2015 NSCJ-J

**National Semiconductor Hong Kong, Ltd.**

1st Floor,  
Cheung Kong Electronic Bldg.  
4 Hing Yip Street  
Kwun Tong  
Kowloon, Hong Kong  
Tel: 3-899235  
Telex: 43866 NSEHK HX  
Cable: NATSEMI HX

**NS Electronics Pty. Ltd.**

Cnr. Stud Rd. & Mtn. Highway  
Bayswater, Victoria 3153  
Australia  
Tel: 03-729-6333  
Telex: AA32096

**National Semiconductor PTE, Ltd.**

10th Floor  
Pub Building, Devonshire Wing  
Somerset Road  
Singapore 0923  
Tel: 652 700047  
Telex: NATSEMI RS 21402

**National Semiconductor Far East, Ltd.**

P.O. Box 68-332 Taipei  
3rd Floor, Apollo Bldg.  
No. 218-7 Chung Hsiao E. Rd.  
Sec. 4 Taipei Taiwan R.O.C.  
Tel: 7310393-4, 7310466-6  
Telex: 22837 NSTW  
Cable: NSTW, TAIPEI

**National Semiconductor Hong Kong, Ltd.**

Korea Liaison Office  
6th Floor, Kunwon Bldg.  
No. 2, 1-GA Mookjung-Dong  
Choong-Ku, Seoul, Korea  
C.P.O. Box 7941 Seoul  
Tel: 267-9473  
Telex: K24942