Publication Number 29-405

MODEL 7/32 PROCESSOR USER'S MANUAL

INTERDATA®
Subsidiary of PERKIN-ELMER
Oceanport, New Jersey 07757, U.S.A.

May 1976

PAGE REVISION STATUS SHEET

PUBLICATION NUMBER 29-405

TITLE 7/32 PROCESSOR USER'S MANUAL

REVISION R00 DATE 6/76

PAGE	REV.	DATE	PAGE	REV.	DATE	PAGE	REV.	DATE
1-1 Thru 1-18	R00	6/76	A2-1 Thru A2-4	R00	6/76			
2-1 Thru 2-52	R00	6/76	A3-1 Thru A3-6	R00	6/76			
3-1 Thru 3-24	R00	6/76	A4-1 Thru A4-2	R00	6/76			
4-1 Thru 4-24	R00	6/76	A5-1 Thru A5-6	R00	6/76			
5-1 Thru 5-36	R00	6/76	A6-1 Thru A6-8	R00	6/76			
6-1 Thru 6-16	R00	6/76	A7-1 Thru A7-4	R00	6/76			
7-1 Thru 7-26	R00	6/76						
8-1 Thru 8-8	R00	6/76						
9-1 Thru 9-10	R00	6/76						
10-1 Thru 10-12	R00	6/76						
A1-1 Thru A1-2	R00	6/76						

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

ILLUSTRATIONS

TABLES

APPENDICES

CHAPTER 1
SYSTEM DESCRIPTION

The Model 7/32 is designed to meet the need for a high performance 32-bit minicomputer. Through the use of 32-bit general registers and a comprehensive instruction set, the Model 7/32 provides fullword data processing power and direct memory addressing up to a limit of one million bytes. The 7/32 System is shown, in block diagram form, in Figure 1-1.

The instruction set includes arithmetic and logical operations, list processing, floating point, cyclic redundancy checking, and bit and byte manipulation. Through this repertoire and direct memory addressing, coding and debugging time is reduced to a minimum.

Two sets of sixteen 32-bit General Registers are provided. Register set selection is controlled by bits in the Program Status Word. Register-to-Register instructions permit operations between any of the 16 registers in the current set, eliminating redundant loads and stores; the multiple register set organization eliminates the overhead incurred in saving and restoring registers when responding to interrupts.

The optional Memory Access Controller (MAC) provides automatic program segmentation, relocation, and protection. The Processor Protect mode enables detection of privileged instructions. These two features are invaluable in process control, data communication, and time-sharing operations to guarantee that a running program cannot interfere with the integrity of the system.

In addition to conventional means of programmed I/O, the Model 7/32 automatically acknowledges all I/O interrupts and performs much of the required overhead prior to activating an Interrupt Service Routine. The Auto Driver Channel can perform data transfers with character translation, longitudinal or cyclic redundancy checking and data buffer chaining without interrupting the running program.

The reader is referred to the following manuals for further information:

Common Assembler Language (CAL) User's Manual, Publication Number 29-375.

ESELCH Programming Manual, Publication Number 29-529.

EDMA Bus Universal Interface Instruction Manual, Publication Number 29-423.

Model 7/32 Maintenance Manual, Publication Number 29-403.

NOTE

Information contained in this manual is subject to design change or product improvement.

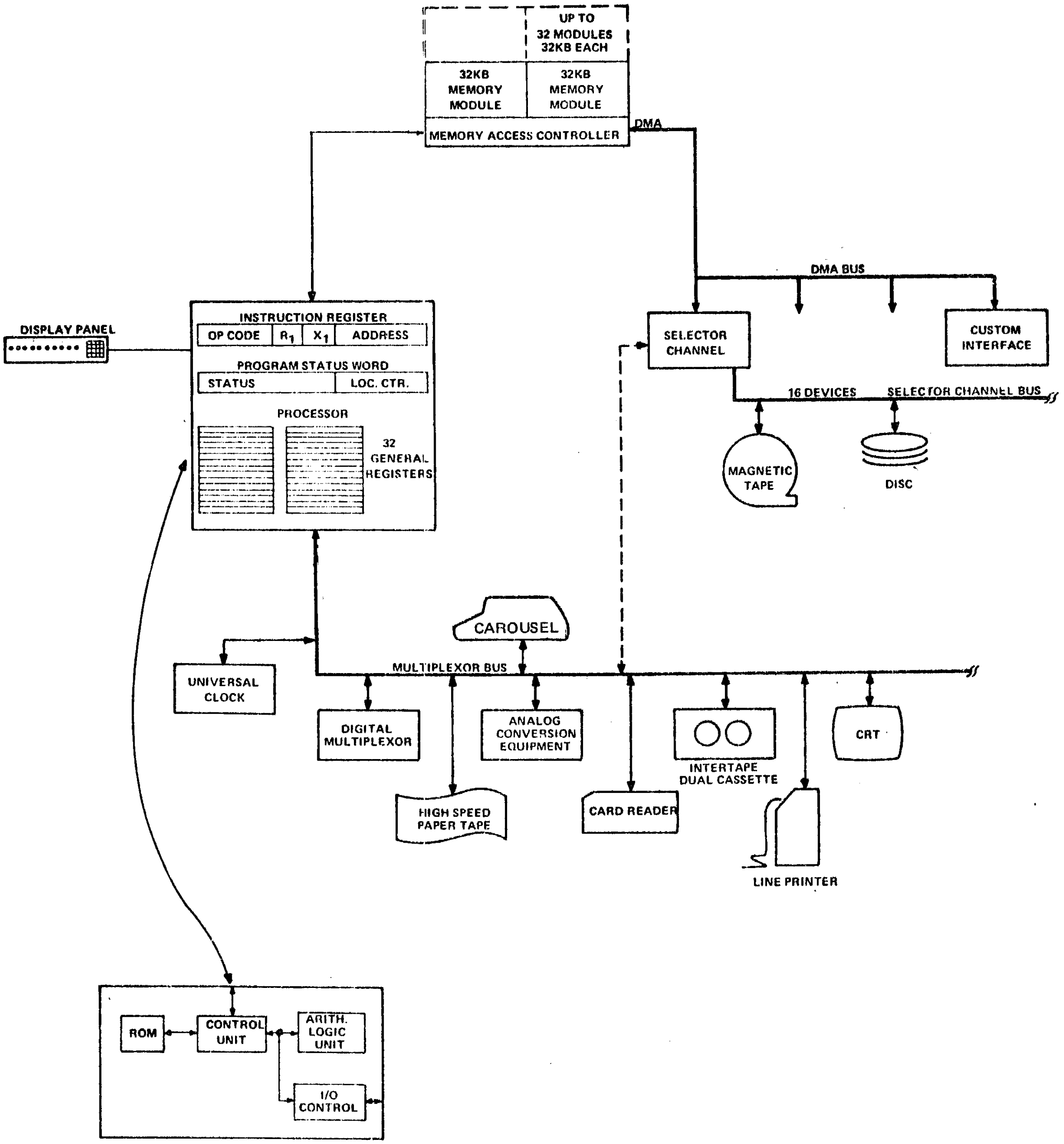


Figure 1-1 Model 7/32 System Block Diagram

The following are major differences between the Model 7/32 and the Model 8/32 Processors from a programmer's point of view:

1. The Model 7/32 Processor has two General Register sets while the Model 8/32 Processor can have two or eight General Register sets depending on the option selected.

2. The Model 7/32 Processor has no I/O Priority Levels while the Model 8/32 Processor can have none or three I/O Priority Levels depending on the option selected.

3. The earlier version of the Model 7/32 Processor has a capability of executing some of the programs written for the INTERDATA 16-Bit Processors. The later version of the Model 7/32 and the Model 8/32 Processors have no such capability.

4. The Model 7/32 Processor does not have an optional Writable Control Store and related instructions as does the Model 8/32 Processor.

5. Fullword operations: In the Model 8/32 Processor, to fetch/store a fullword from/into memory, the fullword data must be aligned on a fullword boundary. This is not the case in the current version of the Model 7/32 Processor. In the Model 7/32, it is sufficient that a fullword data be aligned on a halfword boundary. Thus, a program that executes correctly on the current Model 7/32 may not do so when tried on the Model 8/32. The mnemonics for the instructions that may introduce such a discrepancy are:

   A
   ABL
   AD
   AE
   AM
   ATL
   C
   CD
   CE
   CL
   D
   DD
   DE
   L
   LD
   LE
   LM
   LMD
   LME
   LRA
   M
   MD
   ME
   N
   O
   RB
   RBL
   RTL
   S
   SCP
   SD
   SE
   ST
   STD
   STE
   STM
   STMD
   STME
   SVC
   TLATE
   WB
   X

6. Machine Malfunction Interrupt: In later version of the Model 8/32 Processor, the fullword data read/write on a halfword boundary causes the machine malfunction interrupt to occur, if enabled in the current PSW. After the interrupt is taken, the condition code field of the new PSW is set to 4 (CVGL = 0100).

   In the earlier version of the Model 8/32 Processor fullword data read/write on a halfword boundary forces the address to the fullword boundary and then the data is read/written. Machine Malfunction interrupt does not occur.

   In the current Model 7/32, fullword data read/write on a halfword boundary causes the data to be read from/written into the consecutive halfwords. Machine Malfunction interrupt does not occur.

7. In Model 7/32, the MAC traps 256 bytes. In the Model 8/32, the MAC traps 72 bytes.

8. In Model 7/32, the MAC is optional. In the Model 8/32, the MAC is part of the basic processor.

9. On the average, the Model 8/32 is 2 to 2.5 times faster than the Model 7/32.

10. The Simulate Interrupt (SINT) instruction: on the Model 7/32, the R1 field of the SINT instruction must be zero, specifying that Register Set 0 is to be used.

11. Memory Access Controller interrupt (old Location Counter): In the Model 8/32, it points to the current instruction. In the Model 7/32, it points to the next instruction for data fetch fault or it points to the instruction to be executed in case of execute protect violation.

NOTE

For a detailed description of the Model 8/32 the reader should refer to the Model 8/32 Processor User's Manual, Publication Number 29-428.

PROCESSOR

The Central Processing Unit (CPU), or Processor, controls activities in the system. It executes instructions in a specific sequence and performs arithmetic and logical functions. Included in the Processor's components are:

Program Status Word register
General registers
Floating point registers
Hardware multiply and divide
Floating point hardware

Program Status Word

The 64 bit Program Status Word (PSW) defines the state of the Processor at any given time. (See Figure 1-2.)

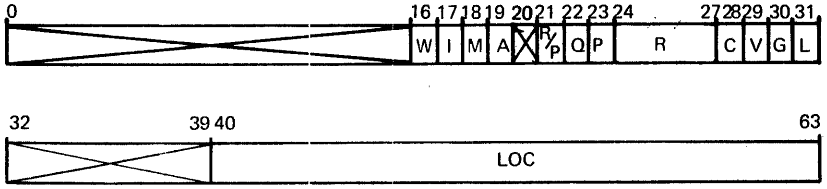


Figure 1-2. Program Status Word

Bits 0:31 are reserved for status information and interrupt masks. Bits 40:63 contain the Location Counter. Unassigned Program Status Word bits must not be used and must always be zero. Status information and interrupt mask bits are defined as follows:

Bit 16	Wait state
Bit 17	Immediate interrupt/ADC Mask
Bit 18	Machine malfunction interrupt mask
Bit 19	Arithmetic fault interrupt mask
Bit 21	Relocation/protection interrupt mask
Bit 22	System queue service interrupt mask
Bit 23	Protect mode
Bits 24:27	Register set select bits
Bits 28:31	Condition Code

Wait State (W)

When this bit is set, the Processor halts normal program execution. It is still responsive to machine malfunction and immediate interrupts, if enabled.

Immediate Interrupt/Auto Driver Channel Mask (I)

Program Status Word Bit 17 controls requests for service from devices on the Multiplexor Bus and Selector Channel. It also controls the Auto Driver Channel. If this bit is set, the Processor responds to the requests. If it is reset, the requests are queued. Refer to Chapter 6 for details of Immediate Interrupt processing.

Machine Malfunction Interrupt Mask (M)

This bit controls interrupts generated when power fails, when power returns, or when parity checking indicates a memory parity error.

Arithmetic Fault Interrupt Mask (A)

This bit controls internal interrupts caused by arithmetic faults: fixed-point quotient overflow or division by zero; or floating point overflow, underflow, or division by zero. If this bit is set, the interrupt is taken. If it is reset, the error condition is ignored.

Relocation Protection Interrupt Mask (R/P)

This bit serves two purposes. It enables the memory access and protect controller (MAC) so that program addresses are automatically relocated. It also enables the relocation/protection interrupt which is generated by the memory access and protect controller. The MAC is optional.

System Queue Service Interrupt Mask (Q)

This bit controls the interrupt generated when the system queue requires service.

Protect Mode (P)

This bit describes an operational state of the Processor. If it is set, the Processor is in the protect mode, and only non-privileged instructions may be executed, to protect the integrity of the system. If this bit is reset, the Processor is in the Supervisor mode, and the currently running program may execute any legal instruction.

Reaister Set Select (R)

The Model 7/32 has two sets of general registers, numbered 0 and 15. Bits 24:27 of the Program Status Word are used to designate the current register set. If Bits 24:27 are all zeroes, register set 0 is selected. If Bits 24:27 are all ones, register set 15 is selected.

Condition Code (CVGL)

Bits 28:31 of the Program Status Word contain the Condition Code. As part of the execution of certain instructions, the state of the Condition Code may be changed to indicate the nature of the result. Not all instructions affect the Condition Code. The state of the Condition Code may be tested with Conditional Branch instructions.

Location Counter (LOC)

The Location Counter controls the sequencing of instruction execution. In normal sequential operation, the Location Counter contains the address of the next instruction to be executed. The instruction is fetched from memory. While the instruction is being executed, the Location Counter is incremented by either two, or four, or six, depending on the length of the instruction. Upon completion of instruction execution, the next instruction is fetched from the location specified by the incremented Location Counter, and the process is repeated.

This sequential mode of operation is altered by Branch instructions, the LPSW and LPSWR, SINT, SVC instructions, and by interrupts. Branch instructions cause the Location Counter to be replaced by a new value derived from the instruction. The LPSW, LPSWR, SINT, and SVC instructions, and interrupts cause the entire Program Status Word to be replaced by a new Program Status Word.

GENERAL REGISTERS

The Model 7/32 has two sets of general registers, numbered 0 and 15. Each register is 32 bits wide. Register set selection is determined by the state of Bits 24:27 of the current Program Status Word. Registers 1 through 15 of any set may be used as index registers. If register 0 is specified, no indexing occurs.

When interrupts occur, the Processor loads pertinent information into preselected registers of the register set 0. The details of this operation are described in Chapter 6. Register set 15, the user set, does not have any specific functional assignments.

Floating Point Registers

There are eight optional single-precision floating point registers, each 32 bits wide. The registers are identified by the even numbers 0 through 14. Floating point operations must always specify the registers with even numbers.

There are eight optional double-precision floating point registers each 64 bits wide. These registers are identified by the even numbers 0 through 14, and are completely separate from the single-precision floating point registers.

Processor Interrupts

Interrupt conditions cause the entire Program Status Word to be replaced by a new Program Status Word, thus breaking the usual sequential flow of instruction execution. When an interrupt condition occurs, the Processor saves its current Program Status Word either in memory or in a pair of general registers of register set 0. It loads information related to the interrupt condition in other registers of this same set. It loads a new Program Status Word from a memory location reserved for the specific interrupt condition. (The immediate interrupt is an exception to the rule. The status portion of the new Program Status Word, Bits 0:31, is forced to a preset value. The Location Counter is loaded from a memory location reserved for the interrupting device. Refer to Chapter 6 for details on interrupt processing.)

Reserved Memory Locations

The following memory locations are reserved for interrupt pointers, Program Status Words, and system constants.

X'000000'	-	X'00001F'	Reserved (Single Precision Floating Point Register, if equipped, Save Area)
X'000020'	-	X'000027'	Machine malfunction interrupt old PSW
X'000028'*	-	X'00002F'	Not used, must be zero
X'000030'	-	X'000037'	Illegal instruction interrupt new PSW
X'000038'	-	X'00003F'	Machine malfunction interrupt new PSW
X'000040'	-	X'000047'	Not used, must be zero
X'000048'	-	X'00004F'	Arithmetic fault interrupt new PSW
X'000050'	-	X'00007F'	Bootstrap loader and device definition table
X'000080'	-	X'000083'	System queue pointer
X'000084'	-	X'000085'	Power Fail PSW save pointer
X'000086'	-	X'000087'	Power Fail Register save pointer
X'000088'	-	X'00008F'	System queue service interrupt new PSW
X'000090'	-	X'000097'	Relocation/protection interrupt new PSW
X'000098'	-	X'00009B'	Supervisor call new PSW status
X'00009C'	-	X'0000BB'	Supervisor call interrupt new PSW location counter values
X'0000BC'	-	X'0000CF'	Not used, must be zero
X'0000D0'	-	X'0002CF'	Interrupt service pointer table
X'0002D0'	-	X'0004CF'	Expanded interrupt service pointer table
X'0004D0'	-	X'0008CF'	Expanded interrupt service pointer table

*Used by Micro-Program

These reserved locations play an important role in both interrupt and input/output processing. For details on these subjects refer to Chapters 6 and 7. In addition to the above, certain locations are reserved for use by the Memory Access Controller. Refer to Chapter 8 for details.

The power down save areas for general registers and PSW must be completely contained within the first 64KB of memory. All new location Counter values are subject to MAC relocation if the new PSW enables MAC (Bit 21 = 1). All other pointers contain absolute addresses not subject to MAC relocation.

Processor Operations

Fixed point arithmetic and logical operations are performed between:

The contents of two fullword registers.

The contents of a fullword register and the contents of a fullword located in memory.

The contents of a fullword register and the contents of a halfword located in memory.

Where the second operand is contained in memory, it may be located in the instruction stream (immediate operation), or it may be located in indexed storage.

In fixed point arithmetic and logical operations between a fullword register and a halfword operand in memory, the halfword operand is expanded to a fullword by propagating the most significant bit into the high order bits before the operation is started. This permits the use of halfword to fullword operations with consistent results, and it provides space economy in that small values do not require fullword locations.

Arithmetic operations on fixed point halfword quantities may produce results that are not entirely consistent with the results that are obtained in a 16-bit Processor. Where this is a problem, the Convert to Halfword Value Register Instruction (CHVR) may be used to adjust the result and the Condition Code so that they are consistent with the same operations in a 16-bit Processor.

Floating point operations take place between the contents of two floating point registers, or between the contents of a floating point register and a floating point operand contained in a fullword or double word in memory. Following floating point operations, the Condition Code is set to indicate the nature of the result.

DATA FORMATS

The Processor performs logical and arithmetic operations on single bits, 8-bit bytes, 16-bit halfwords, 32-bit fullwords, and 64-bit double words. This data may represent a fixed point number, a floating point number, or logical information.

Fixed Point Data

Fixed point arithmetic operands may be either 16-bit halfwords or 32-bit fullwords. In fullword multiply and divide operations, 64-bit operands are manipulated. Fixed point data are treated as 15-bit signed integers in the halfword format, and as 31-bit signed integers in the fullword format. Positive numbers are expressed in true binary form with a Sign bit of zero. Negative numbers are represented in two's complement form with a Sign bit of one. The numerical value of zero is represented with all bits zero. Refer to Chapter 4 for details on fixed point data representation.

Floating Point Data

A floating point number consists of a signed exponent and a signed fraction. The quantity expressed by this number is the product of the fraction and the number 16 raised to the power represented by the exponent. Each floating point value requires a 32-bit fullword or a 64-bit double word, of which eight bits are used for the sign and exponent, and the remaining bits are used for the fraction. Refer to Chapter 5 for details on floating point data representation.

Logical Data

Logical operations manipulate 8-bit bytes, 16-bit halfwords, and 32-bit fullwords. In addition, it is possible to perform logical operations on single bits located in bit arrays. Refer to Chapter 2 for details on logical data representation.

INSTRUCTION FORMATS

The INTERDATA instruction formats provide a concise method of representing required operations for easy interpretation by the Processor. There are seven basic formats, shown in Figure 1-4. The abbreviations used in the figure have the following meanings:

OP	Operation code
R1	First operand register
R2	Second operand register
N	A four bit immediate value
X2	Second operand single index register
D2	Second operand displacement
FX2	Second operand first index register
SX2	Second operand second index register
A2	Second operand direct address
I2	Second operand immediate value

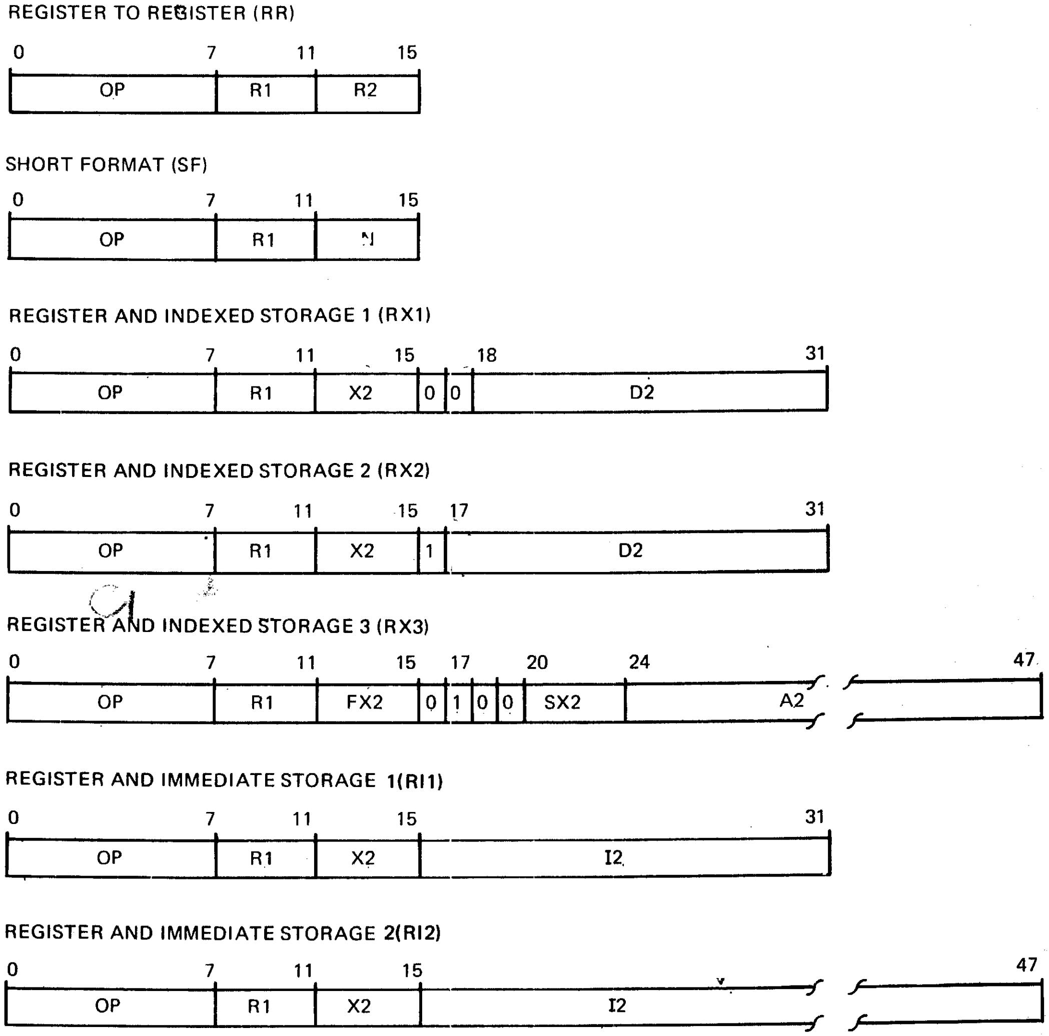

REGISTER TO REGISTER (RR)
0 7 11 15
OP R1 R2
SHORT FORMAT (SF)
0 7 11 15
OP R1 N
REGISTER AND INDEXED STORAGE 1 (RX1)
0 7 11 15 18 31
OP R1 X2 0 0 D2
REGISTER AND INDEXED STORAGE 2 (RX2)
0 7 11 15 17 31
OP R1 X2 1 D2
REGISTER AND INDEXED STORAGE 3 (RX3)
0 7 11 15 17 20 24 47
OP R1 FX2 0 1 0 0 SX2 A2
REGISTER AND IMMEDIATE STORAGE 1(RI1)
0 7 11 15 31
OP R1 X2 I2
REGISTER AND IMMEDIATE STORAGE 2(RI2)
0 7 11 15 47
OP R1 X2 I2

Figure 1-4. Instruction Formats

Most instructions in the extended series may be expressed in two or more formats. This feature provides flexibility in data organization and instruction sequencing.

When working with the Interdata Common Assembler Language (CAL) assembler, it is not necessary to specify the instruction format explicitly. The assembler chooses the most economical format and supplies the required bits in the machine code. When double indexing is implied, the assembler always chooses the RX3 format.

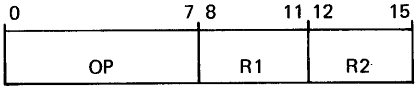
Branch Instruction Formats

The Branch instructions use the RR, SF, and all variations on the RX formats. However, in the Conditional Branch instructions, the R1 field does not specify a register. Instead, it contains a mask value (labelled M1 in the instruction descriptions), which is tested with the Condition Code. The INTERDATA CAL assembler provides a series of Extended Branch Mnemonics which make it possible to specify a Conditional Branch without specifying the mask value explicitly. For a summary of the Extended Branch Mnemonics, see Appendix 4.

Programming Examples

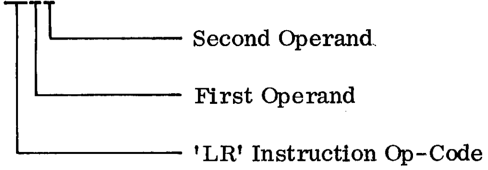
Each of the following programming examples refers to the sample assembly language program shown in Figure 1-4. Note the use of symbolic equates for general registers. Machine code generated and the result of each instruction are dependent upon the physical and logical placement of the instructions, respectively.

Register to Register (RR) Format

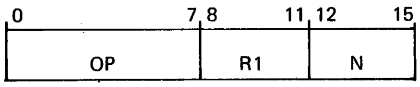


In this 16 bit format, Bits 0:7 contain the operation code. Bits 8:11 contain the R1 field, and Bits 12:15 contain the R2 field. In most RR instructions, the register specified by R1 contains the first operand, and the register specified by R2 contains the second operand. For example:

Machine Code	Label	Assembler Notation
0865	RR	LR R6, R5

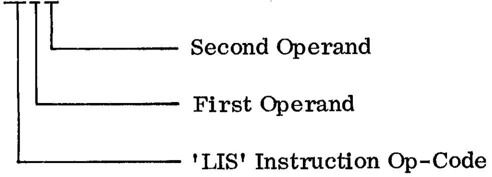


Short Form (SF) Format



This 16-bit format provides space economy when working with small values. Bits 0:7 contain the operation code. Bits 8:11 contain the R1 field. Bits 12:15 contain the N field. In arithmetic and logical operations, the register specified by R1 contains the first operand. The N field contains a four bit immediate value (0:15) used as the second operand. For example:

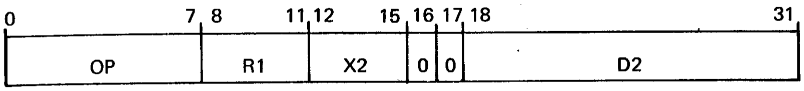
Machine Code	Label	Assembler Notation
245E	SF	LIS R5,14



```
                           1         SCRAT
                           2         TARGT   32
                           3         NORX3
                           4         WIDTH   120
000000I                    5         NOSQZ
                           6   *
                           7   *
                           8   *
           0000 0005       9   R5        EQU     5                          GENERAL REGISTER 5
           0000 0006      10   R6        EQU     6                          GENERAL REGISTER 6
           0000 0007      11   R7        EQU     7                          GENERAL REGISTER 7
           0000 0008      12   R8        EQU     8                          GENERAL REGISTER 8
           0000 0009      13   R9        EQU     9                          GENERAL REGISTER 9
           0000 000A      14   R10       EQU     10                         GENERAL REGISTER 10
           0000 000B      15   R11       EQU     11                         GENERAL REGISTER 11
                          16   *
000000I    245E           17   SF        LIS     R5,14                      (R5) = Y'0000000E'
                          18   *
000002I    0865           19   RR        LR      R6,R5                      (R6) = Y'0000000E'
                          20   *
000004I    4050 1000      21   RX1.EX1   STH     R5,X'1000'                 (X'1000') = X'000E'
                          22   *
000008I    4056 0FF2      23   RX1.EX2   STH     R5,X'0FF2'(R6)             (X'1000') = X'000E'
                          24   *
00000CI    4050 8004      25   RX2.EX1   STH     R5,LOC1                    (LOC1) = X'000E'
                          26   *
000010I    4300 8004      27             B       RI1.EX1
000014I    0000 0000      28   LOC1      DC      F'0'
                          29   *
000018I    CA90 8000      30   RI1.EX1   LHI     R9,X'8000'                 (R9) = Y'FFFF8000'
                          31   *
00001CI    C895 8000      32   RI1.EX2   LHI     R9,X'8000'(R5)             (R9) = Y'FFFF800E'
                          33   *
000020I    F8A0 0000 8000 34   RI2.EX1   LI      R10,X'8000'                (R10) = Y'00008000'
                          35   *
000026I    F8BA 0001 7FFE 36   RI2.EX2   LI      R11,Y'17FFE'(R10)          (R11) = Y'0001FFFE'
                          37   *
00002CI    4050 FFE4      38   RX2.EX2   STH     R5,LOC1                    (LOC1) = X'000E'
                          39   *
000030I    4056 FFD2      40   RX2.EX3   STH     R5,LOC1-14(R6)             (LOC1) = X'000E'
                          41   *
000034I    5870 4001 0000 42   RX3.EX1   L       R7,Y'10000'                (R7) = (Y'10000')
                          43   *
00003AI    5885 4601 FFE4 44   RX3.EX2   L       R8,Y'20000'-28(R5,R6)      (R8) = (Y'20000')
                          45   *
000040I    4300 FFBC      46             B       SF
                          47   *
000044I                   48             END
```

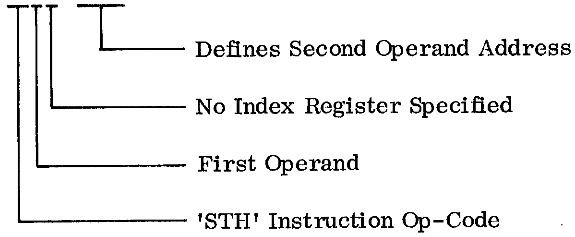
Figure 1-5. 32-Bit Instruction Format Examples

Register and Indexed Storage One (RX1) Format

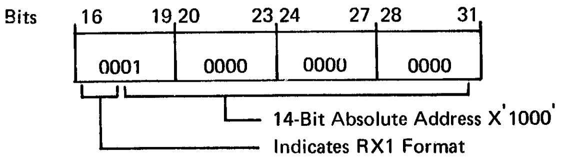


This is a 32-bit format in which Bits 0:7 contain the operation code, Bits 8:11 contain the R1 field, Bits 12:15 contain the X2 field, Bits 16 and 17 must be zero, and Bits 18:31 contain the D2 field. In general, the register specified by R1 contains the first operand. The second operand is located in memory at the address obtained by adding the contents of the second operand index register, specified by X2, and the 14-bit absolute address contained in the D2 field. For example:

Machine Code	Label	Assembler Notation
4050 1000	RX1.EX1	STH R5,X'1000'

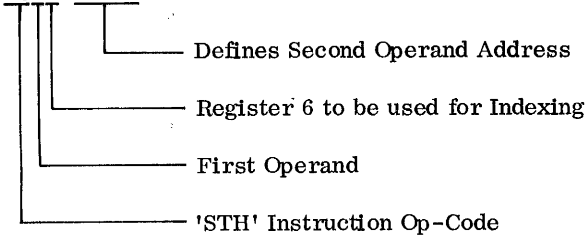

Defines Second Operand Address
No Index Register Specified
First Operand
'STH' Instruction Op-Code

The Second Operand address is calculated as follows:

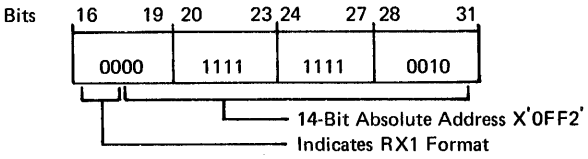


No indexing is specified. Therefore, the second operand address is X'1000'.

Machine Code	Label	Assembler Notation
4056 0FF2	RX1.EX2	STH R5,X'0FF2'(R6)


Defines Second Operand Address
Register 6 to be used for Indexing
First Operand
'STH' Instruction Op-Code

The Second Operand address is calculated as follows:



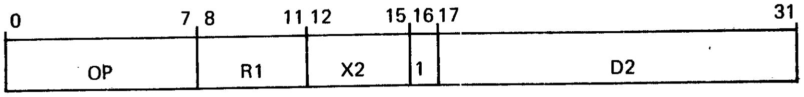
Second Operand Address

= contents of D2 field + contents of the Index Register 6 (see Figure 1-5)

= X'0FF2' + Y'0000000E'

= Y'00001000'

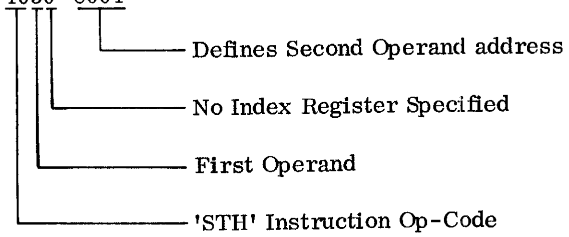
Register and Indexed Storage Two (RX2) Format



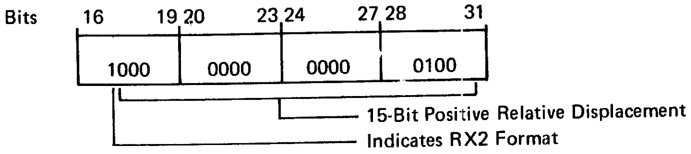
This format provides relative addressing capability in a 32 bit instruction word. Bits 0:7 contain the operand code. Bits 8:11 contain the R1 specification. Bits 12:15 contain the X2 specification. Bit 16 must always be one. Bits 17:31 contain the relative displacement, D2.

In the RX2 format, the register specified by R1 contains the first operand. The address of the second operand, in memory, is calculated by adding the value contained in the incremented location counter (the address of the next sequential instruction) and the sum of (1) the 32-bit representation of the 15-bit signed number contained in the D2 field, and (2) the contents of the index register specified by X2. Negative numbers in the D2 field are expressed in two's complement notation. For example:

Machine Code	Label	Assembler Notation
4050 8004	RX2.EX1	STH R5,LOC1


Defines Second Operand address
No Index Register Specified
First Operand
'STH' Instruction Op-Code

The Second Operand address is calculated as follows:



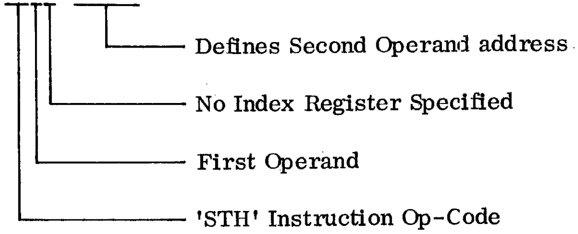
Second Operand Address

= 32-bit Expansion of contents of D2 field + contents of incremented Location Counter (see Figure 1-5).

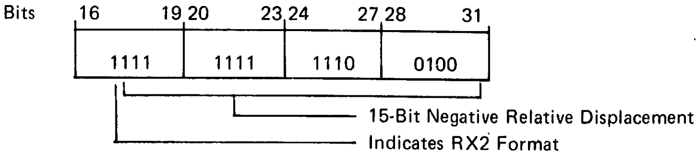
= Y'00000004' + Y'00000010'

= Y'00000014'

Machine Code	Label	Assembler Notation
4050 FFE4	RX2.EX2	STH R5,LOC1


Defines Second Operand address
No Index Register Specified
First Operand
'STH' Instruction Op-Code

The Second Operand address is calculated as follows:



Second Operand Address

= 32-bit Expansion of contents of D2 field + contents of incremented Location Counter (see Figure 1-5).

= Y'FFFFFFE4' + Y'00000030'

= Y'00000014'

Machine Code	Label	Assembler Notation
4056 FFD2	RX2.EX3	STH R5,LOC1-14(R6)

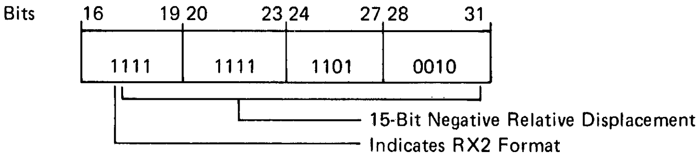
Defines Second Operand address

Register 6 to be used for Indexing

First Operand

'STH' Instruction Op-Code

The Second Operand address is calculated as follows:



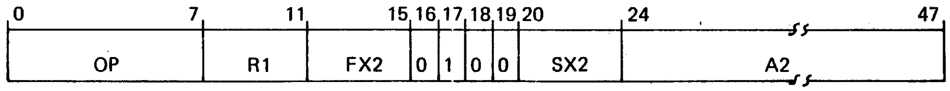
Second Operand Address

= 32-Bit Expansion of D2 field + contents of incremented Location Counter + contents of Index Register 6 (See Figure 1-5).

= Y'FFFFFFDE' + Y'00000034' + Y'0000000E'

= Y'00000014'

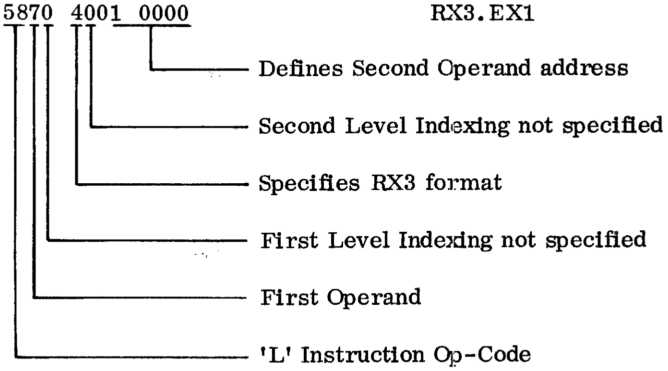
Register and Indexed Storage Three (RX3) Format



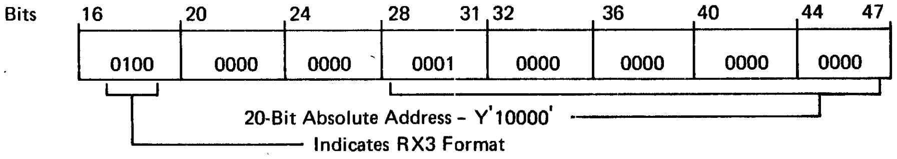
This is a 48-bit format in which double indexing is permitted. Bits 0:7 contain the operation code. Bits 8:11 contain the R1 specification. Bits 12:15 contain the first index specification, FX2. Bit 16 must be zero. Bit 17 must be one. Bits 18:19 must be zero. Bits 20:23 contain the second index specification, SX2. Bits 24:47 contain a 24-bit address, A2. Second level indexing is allowed even if first level indexing is not specified.

In general, the first operand is contained in the register specified by R1. The second operand is located in memory. Its memory address is obtained by adding the contents of the first index register and the contents of the second index register, and then adding to this result the contents of the A2 field. For example:

Machine Code	Label	Assembler Notation
5870 4001 0000	RX3.EX1	L R7, Y'10000'


Defines Second Operand address
Second Level Indexing not specified
Specifies RX3 format
First Level Indexing not specified
First Operand
'L' Instruction Op-Code

The Second Operand address is calculated as follows:

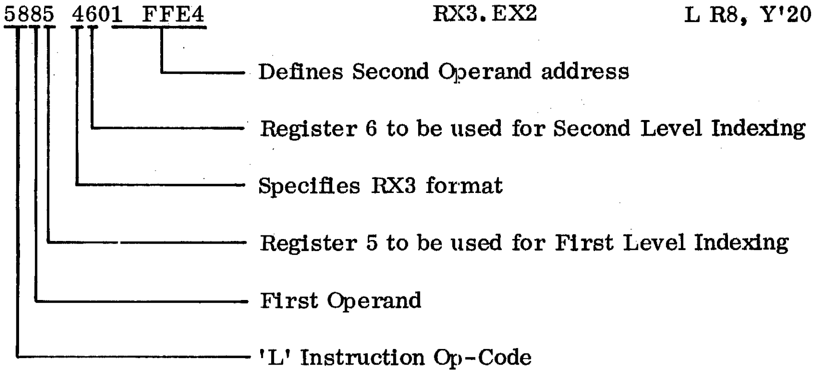


Second Operand Address

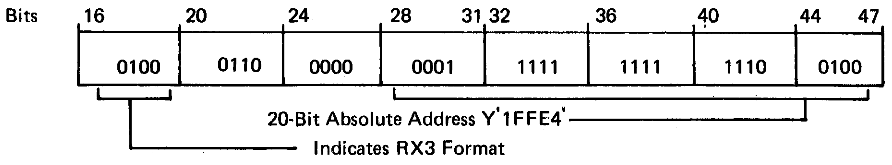
= Contents of A2 field

= Y'00010000'

Machine Code	Label	Assembler Notation
5885 4601 FFE4	RX3.EX2	L R8, Y'20000'-28(R5,R6)


Defines Second Operand address
Register 6 to be used for Second Level Indexing
Specifies RX3 format
Register 5 to be used for First Level Indexing
First Operand
'L' Instruction Op-Code

The Second Operand address is calculated as follows:



Second Operand Address

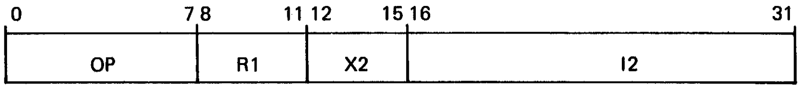
= contents of A2 field + contents of Index Register 6

+ contents of Index Register 5 (see Figure 1-5).

= Y'0001FFE4' + Y'0000000E' + Y'0000000E'

= Y'00020000'

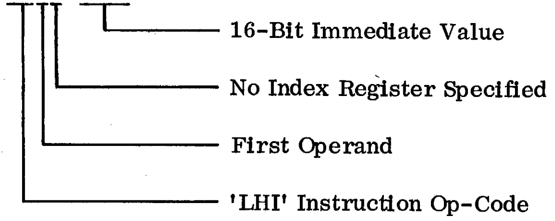
Register and Immediate Storage One (RI1) Format



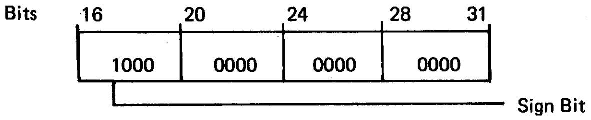
This format represents a 32-bit instruction word. Bits 0:7 contain the operation code. Bits 8:11 contain the R1 specification. Bits 16:31 contain the 16 bit immediate value, I2.

In this format, the register specified by R1 contains the first operand. The 32-bit effective second operand is obtained by adding together the 32-bit representation of the signed 16-bit value contained in the I2 field, and the contents of the register specified by X2. For example:

Machine Code	Label	Assembler Notation
C890 8000	RI1.EX1	LHI R9,X'8000'



The Second Operand is calculated as follows:

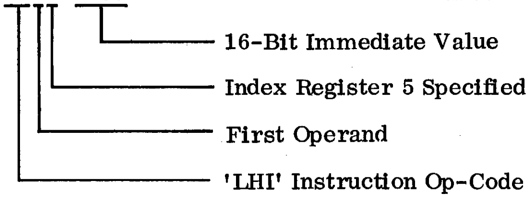


Second Operand

= 32-Bit representation of X'8000'

= Y'FFFF8000'

Machine Code	Label	Assembler Notation
C895 8000	RI1.EX2	LHI R9,X'8000'(R5)



The Second Operand is calculated as follows:

Bits	16	20	24	27 31
	1000	0000	0000	0000

Sign Bit

Second Operand

= 32-Bit representation of X'8000' + the contents of Index Register 5 (See Figure 1-5).

= Y'FFFF8000' + Y'0000000E'

= Y'FFFF800E'

Register and Immediate Storage Two (RI2) Format

0 7	11	15	47
OP	R1	X2	I2

This is a 48-bit instruction format. Bits 0:7 contain the operation code. Bits 8:11 contain the R1 specification. Bits 12:15 contain the X2 specification. Bits 16:47 contain the 32-bit immediate value, I2.

The first operand is contained in the register specified by R1. The second operand is obtained by adding the contents of the index register, specified by X2, and the 32 bit immediate value contained in the I2 field. For example:

Machine Code	Label	Assembler Notation
F8A0 0000 8000	RI2.EX1	LI R10,X'8000'

32-Bit Immediate Field

No Index Register Specified

First Operand

'LI' Instruction Op-Code

The Second Operand is calculated as follows:

Bits	16	20	24	28	32	36	40	44 47
	0000	0000	0000	0000	1000	0000	0000	0000

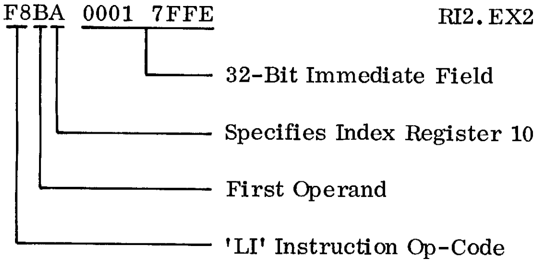
32-Bit Immediate Value

Second Operand

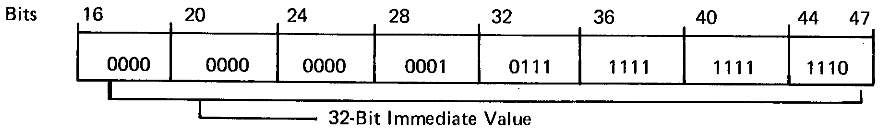
= Contents of I2 Field

= Y'00008000'

Machine Code	Label	Assembler Notation
F8BA 0001 7FFE	RI2.EX2	LI R11,Y'17FFE'(R10)



The Second Operand is calculated as follows:



Bits	16	20	24	28	32	36	40	44 47
	0000	0000	0000	0001	0111	1111	1111	1110

Second Operand

= Contents of I2 Field + contents of Index Register 10 (See Figure 1-5).

= Y'00017FFE' + Y'00008000'

= Y'0001FFFE'

CHAPTER 2
LOGICAL OPERATIONS

The set of logical instructions provides a means for the manipulation of binary data. Many of the instructions grouped with the logical set may also be used in arithmetic and other operations. These instructions include loads, stores, compares, shifts, list processing, translation, and cyclic redundancy checks.

DATA FORMATS

Logical data may be organized as bytes, halfwords, fullwords, or bit arrays of up to 2^{31} bits as shown in Figure 2-1.

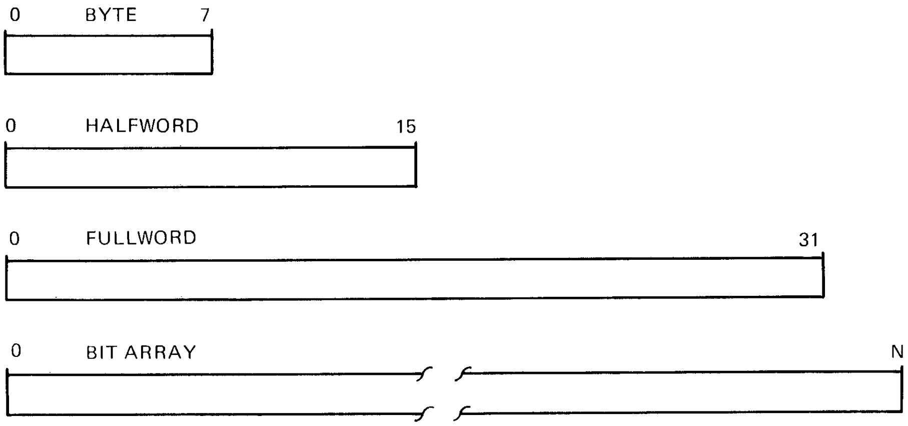


Figure 2-1. Logical Data

OPERATIONS

In logical operations between the contents of a general register and a halfword operand, the halfword operand is expanded to a fullword before the operation starts. The halfword is expanded by propagating the most significant bit through Bits 15:0 of the fullword.

Boolean Operations

The Boolean operators AND, OR, and Exclusive OR (XOR) operate on halfword and fullword quantities. All bits in both operands participate individually. The Boolean functions are defined as follows:

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0 (logical product)
1 AND 1 = 1

0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1 (logical sum)
1 OR 1 = 1

0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1 (logical difference)
1 XOR 1 = 0

Translation

The translate instruction is used to translate a character directly, or to effect an unconditional branch to a special translate subroutine. Associated with the translate instruction is a translation table. The entries in the table are halfwords as shown in Figure 2-2.

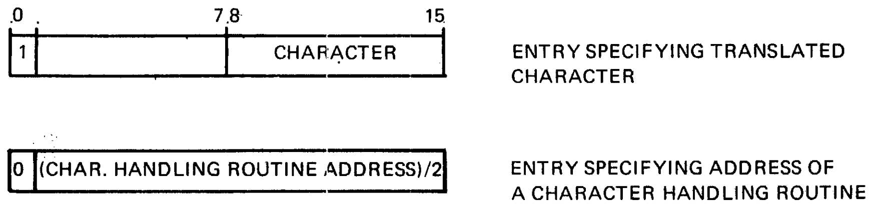

0
7
8
15
1
CHARACTER
ENTRY SPECIFYING TRANSLATED CHARACTER
0
(CHAR. HANDLING ROUTINE ADDRESS)/2
ENTRY SPECIFYING ADDRESS OF A CHARACTER HANDLING ROUTINE

Figure 2-2. Translation Table Entry

The character to be translated is a byte of logical data. This unsigned quantity is doubled and used as an index into the table. If the corresponding entry has a one in bit Position zero, then Bits 8:15 contain the character to be substituted for the data character. If there is a zero in bit Position zero, then Bits 1:15 contain the address, divided by two, of the translate routine. When the translate instruction results in a branch, this value is doubled to produce the address of the routine. Because this result is a 16 bit address, the software routine must be located in the first 64KB of the program. (The program may reside anywhere in memory if it is relocated by the relocation and protection module.) The translate table may contain up to 256 entries. However, if the data characters are always less than eight bits, fewer entries are required.

List Processing

The list processing instructions manipulate a circular list as defined in Figure 2-3.

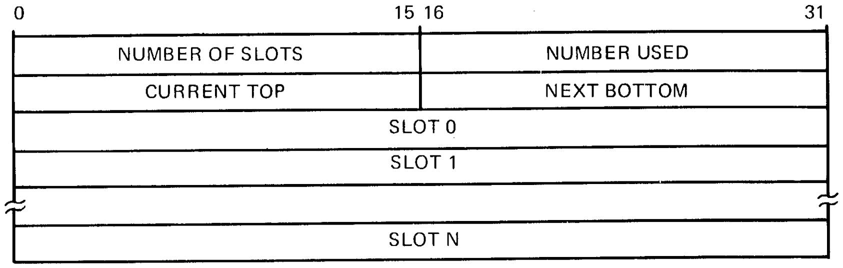


Figure 2-3. Circular List Definition

The first four halfwords contain the list parameters. Immediately following the parameter block is the list itself. The first fullword in the list is designated Slot 0. The remaining slots are designated 1, 2, 3, etc., up to a maximum slot number which is equal to the number in the list minus one. An absolute maximum of 65,535 fullword slots may be specified. (Slots are designated 0 through X'FFFE'.)

The first parameter halfword indicates the number of slots (fullwords) in the entire list. The second parameter halfword indicates the current number of slots being used. When this halfword equals zero, the list is empty. When this halfword equals the number of slots in the list, the list is full. Once initialized, this halfword is maintained automatically. It is incremented when elements are added to the list and decremented when elements are removed.

The third and fourth halfwords of the list parameter block specify the current top of the list and the next bottom of the list respectively. These pointers are also updated automatically. See Figure 2-4.

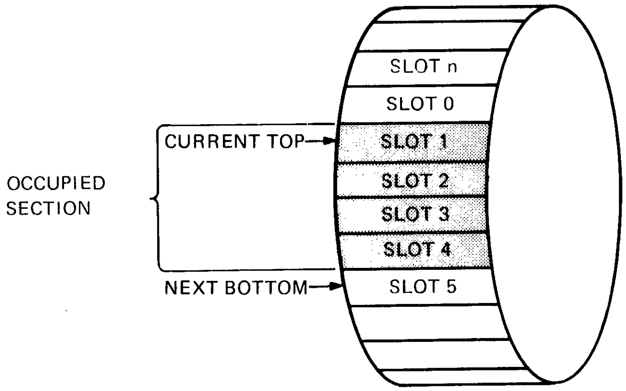


Figure 2-4. Circular List

LOGICAL INSTRUCTION FORMATS

The logical instructions use the Register to Register (RR), the Register and Indexed Storage (RX), and the Register and Immediate Storage (RI) instruction formats.

LOGICAL INSTRUCTIONS

The instructions described in this section are:

L	Load
LR	Load Register
LI	Load Immediate
LIS	Load Immediate Short
LCS	Load Complement Short
LH	Load Halfword
LHI	Load Halfword Immediate
LA	Load Address
LRA	Load Real Address
LHL	Load Halfword Logical
LM	Load Multiple
LB	Load Byte
LBR	Load Byte Register
EXHR	Exchange Halfword Register
EXBR	Exchange Byte Register
ST	Store
STH	Store Halfword
STM	Store Multiple
STB	Store Byte
STBR	Store Byte Register
CL	Compare Logical
CLR	Compare Logical Register
CLI	Compare Logical Immediate
CLH	Compare Logical Halfword
CLHI	Compare Logical Halfword Immediate
CLB	Compare Logical Byte
N	AND
NR	AND Register
NI	AND Immediate
NH	AND Halfword
NHI	AND Halfword Immediate
O	OR
OR	OR Register
OI	OR Immediate
OH	OR Halfword
OHI	OR Halfword Immediate
X	Exclusive OR
XR	Exclusive OR Register
XI	Exclusive OR Immediate
XH	Exclusive OR Halfword
XHI	Exclusive OR Halfword Immediate
TI	Test Immediate
THI	Test Halfword Immediate
SLL	Shift Left Logical
SLLS	Shift Left Logical Short
SRL	Shift Right Logical
SRLS	Shift Right Logical Short
SLHL	Shift Left Halfword Logical
SLHLS	Shift Left Halfword Logical Short
SRHL	Shift Right Halfword Logical
SRHLS	Shift Right Halfword Logical Short
RLL	Rotate Left Logical
RRL	Rotate Right Logical
TS	Test and Set
TBT	Test Bit
SBT	Set Bit
CBT	Complement Bit
RBT	Reset Bit
CRC12	Cyclic Redundancy Check Modulo 12
CRC16	Cyclic Redundancy Check Modulo 16
TLATE	Translate
ATL	Add to Top of List
ABL	Add to Bottom of List
RTL	Remove from Top of List
RBL	Remove from Bottom of List

INSTRUCTIONS

Load (L)
Load Register (LR)
Load Immediate (LI)
Load Immediate Short (LIS)
Load Complement Short (LCS)

Assembler Notation		Op-Code	Format
L	R1,D2 (X2)	58	RX1, RX2
L	R1,A2 (FX2,SX2)	58	RX3
LR	R1,R2	08	RR
LI	R1,I2 (X2)	F8	RI2
LIS	R1,N	24	SF
LCS	R1,N	25	SF

Operation

The second operand replaces the contents of the register specified in R1.

Condition Code

C	V	G	L	
0	0	0	0	Value is ZERO
0	0	0	1	Value is not ZERO
0	0	1	0	Value is not ZERO

Programming Note

The Load Immediate Short instruction causes the four bit second operand to be expanded to a 32 bit fullword with high order bits forced to ZERO. This fullword replaces the contents of the register specified by R1.

The Load Complement Short instruction causes the four bit second operand to be expanded to a 32 bit fullword with high order bits forced to ZERO. The two's complement value of this fullword replaces the contents of the register specified by R1.

When the Load instructions operate on fixed point data, the Condition Code indicates ZERO (no flags), negative (L flag), or positive (G flag) value.

In the RR format, if R1 equals R2, the Load instruction functions as a test on the contents of the register.

In the RX formats, the second operand must be located on a fullword boundary.

Example LCS

Assembler Notation	Machine Code	Comments
LCS REG8, 7	2587	LOAD -7 INTO REG8

Result of LCS Instruction

(REG8) = FFFF FFF9

Condition Code = 0001 (L = 1)

INSTRUCTIONS

Load Halfword (LH)
Load Halfword Immediate (LHI)

Assembler Notation		Op-Code	Format
LH	R1, D2 (X2)	48	RX1, RX2
LH	R1, A2 (FX2, SX2)	48	RX3
LHI	R1, I2 (X2)	C8	RI1

Operation

The halfword second operand is expanded to a fullword by propagating the most significant bit through Bits 15:0. This fullword replaces the contents of the register specified by R1.

Condition Code

C	V	G	L	
0	0	0	0	Value is ZERO
0	0	0	1	Value is not ZERO
0	0	1	0	Value is not ZERO

Programming Note

When the Load Halfword instructions operate on fixed point data, the Condition Code indicates zero (no flags), negative (L flag), or positive (G flag) value.

In the RX formats, the second operand must be located on a halfword boundary.

INSTRUCTION

Load Address (LA)

Assembler Notation		Op-Code	Format
LA	R1,D2 (X2)	E6	RX1, RX2
LA	R1,A2 (FX2,SX2)	E6	RX3

Operation

The effective address of the second operand (24 bits) replaces Bits 8:31 of the register specified by R1. Bits 0:7 of the register specified by R1 are forced to ZERO.

Condition Code

Unchanged

Programming Note

The length of the address quantity depends on the internal structure of the particular machine. Thus, in a Processor with a maximum address length of 20 bits, the calculated address replaces bits 12:31 of the register specified by R1, and bits 0:11 are forced to ZERO. In a Processor with maximum address length of 24 bits, the calculated address replaces bits 8:31 of the register specified by R1, and bits 0:7 are forced to ZERO.

INSTRUCTION

Load Real Address (LRA)

Assembler Notation		Op-Code	Format
LRA	R1, D2(X2)	63	RX1, RX2
LRA	R1, A2(FX2, SX2)	63	RX3

Operation

This instruction simulates the operation of a memory access controller. The register specified by R1 contains a program address (not relocated). The second operand address points to a relocation/protection module parameter block.

The address contained in the register specified by R1 is relocated, using the appropriate parameters. The relocated address replaces the contents of the register specified by R1.

Condition Code

C	V	G	L	
0	0	0	0	No restrictions
0	0	0	1	Not executable
0	0	1	0	Not writable
0	1	0	0	Not present
1	0	0	0	Not mapped (Limit violation)

Programming Note

If the address is not mapped or not present, the register specified by R1 is unchanged.

The second operand location must specify a fullword boundary.

This instruction is supported by the new Model 7/32 microcode. It is therefore not supported in all the models.

Example: LRA

This example performs an address translation in the same manner as the MAC.

For this example, Register 1 contains X'54341', MACREG is the starting address of a copy of the MAC Registers. The fifth fullword entry located at MACREG+X'14' contains X'0FF24170'.

Assembler Notation	Machine Code	Comments
LRA REG1, MACREG	6310 8100	The first digit of the program address (5) is used to index into MACREG

Result of LRA Instruction

(REG1) = 28441 (24100 + 04341)
MACREG = Unchanged
Condition Code = 0010 (not writable)

INSTRUCTION

Load Halfword Logical (LHL)

Assembler Notation		Op-Code	Format
LHL	R1,D2 (X2)	73	RX1,RX2
LHL	R1,A2 (FX2,SX2)	73	RX3

Operation

The halfword second operand replaces Bits 16:31 of the register specified by R1. Bits 0:15 of the register specified by R1 are forced to ZERO.

Condition Code

C	V	G	L	
0	0	0	0	Value is ZERO
0	0	1	0	Value is not ZERO

Programming Note

The second operand must be located on a halfword boundary.

INSTRUCTION

Load Multiple (LM)

Assembler Notation		Op-Code	Format
LM	R1,D2 (X2)	D1	RX1,RX2
LM	R1,A2 (FX2,SX2)	D1	RX3

Operation

Successive registers, starting with the register specified by R1, are loaded from successive memory locations, starting with the location specified as the effective address of the second operand. Each register is loaded with a fullword from memory. The process stops when Register 15 has been loaded.

Condition Code

Unchanged

Programming Note

The second operand must be located on a fullword boundary.

INSTRUCTIONS

Load Byte (LB)
Load Byte Register (LBR)

Assembler Notation		Op-Code	Format
LB	R1,D2 (X2)	D3	RX1,RX2
LB	R1,A2 (FX2,SX2)	D3	RX3
LBR	R1,R2	93	RR

Operation

The eight-bit second operand replaces the least significant bits (Bits 24:31) of the register specified by R1. Bits 0:23 of the register are forced to ZERO.

Condition Code

Unchanged

Programming Note

In the Load Byte Register instruction, the second operand is taken from the least significant eight bits (Bits 24:31) of the register specified by R2.

INSTRUCTION

Exchange Halfword Register (EXHR)

Assembler Notation	Op-Code	Format
EXHR R1,R2	34	RR

Operation

Bits 0:15 of the register specified by R2 replace Bits 16:31 of the register specified by R1. Bits 16:31 of the register specified by R2 replace Bits 0:15 of the register specified by R1.

Condition Code

Unchanged

Programming Note

If R1 equals R2, the two halfwords contained within the register are exchanged.
If R1 does not equal R2, the contents of R2 are unchanged.

Example: EXHR

Assembler Notation	Machine Code	Comments
LI REG5, Y'0ABCDEF9'	F850 0ABC DEF9	(REG 5) = 0ABCDEF9
LI REG7, Y'12345678'	F870 1234 5678	(REG 7) = 12345678.
EXHR REG5,REG7	3457	

Result of EXHR Instruction

(REG 5) = 56781234
(REG 7) = 12345678
Condition Code = Unchanged

INSTRUCTION

Exchange Byte Register (EXBR)

Assembler Notation	Op-Code	Format
EXBR R1,R2	94	RR

Operation

The two eight-bit bytes contained in Bits 16:31 of the register specified by R2 are exchanged and loaded into Bits 16:31 of the register specified by R1. Bits 0:15 of the register specified by R1 are unchanged. The register specified by R2 is unchanged.

Condition Code

Unchanged

Programming Note

R1 and R2 may specify the same register. In this case, the two bytes in Bits 16:31 of the register specified by R2 are exchanged.

Example: EXBR

Assembler Notation	Machine Code	Comments
LI REG7, X'5A6B3C4D'	F870 5A6B 3C4D	(REG7) = 5A6B3C4D
LI REG3, Y'98761234'	F830 9876 1234	(REG3) = 98761234
EXBR REG7,REG3	9473	

Result of EXBR Instruction

(REG7) = 5A6B3412
(REG3) = 98761234
Condition Code = Unchanged

INSTRUCTION

Store (ST)

Assembler Notation		Op-Code	Format
ST	R1, D2 (X2)	50	RX1, RX2
ST	R1, A2 (FX2, SX2)	50	RX3

Operation

The 32 bit contents of the register specified by R1 replace the contents of the memory location specified by the effective address of the second operand.

Condition Code

Unchanged

Programming Note

The second operand location must be on a fullword boundary.

INSTRUCTION

Store Halfword (STH)

Assembler Notation		Op-Code	Format
STH	R1,D2 (X2)	40	RX1,RX2
STH	R1,A2 (FX2,SX2)	40	RX3

Operation

Bits 16:31 of the register specified by R1 replace the contents of the memory location specified by the effective address of the second operand.

Condition Code

Unchanged

Programming Note

The second operand location must be on a halfword boundary.

INSTRUCTION

Store Multiple (STM)

Assembler Notation		Op-Code	Format
STM	R1, D2 (X2)	D0	RX1, RX2
STM	R1, A2 (FX2, SX2)	D0	RX3

Operation

The fullword contents of registers, starting with the register specified by R1, replace the contents of successive memory locations, starting with the location specified by the effective address of the second operand. The process stops when Register 15 has been stored.

Condition Code

Unchanged

Programming Note

The second operand location must be on a fullword boundary.

INSTRUCTIONS

Store Byte (STB)
Store Byte Register (STBR)

Assembler Notation		Op-Code	Format
STB	R1, D2 (X2)	D2	RX1, RX2
STB	R1, A2 (FX2, SX2)	D2	RX3
STBR	R1, R2	92	RR

Operation

The least significant eight bits (Bits 24:31) of the register specified by R1 are stored in the second operand location.

Condition Code

Unchanged

Programming Note

In the Store Byte Register instruction, the eight bit quantity is stored in Bits 24:31 of the register specified by R2. Bits 0:23 of the register are unchanged.

Example: STBR

```
Assembler Notation           Machine Code      Comments

LI   REG4, Y'13577531'       F840 1357 7531    (REG4) = 13577531
LI    REG3, Y'24688642'      F830 2468 8642    (REG3) = 24688642
.
.
.
STBR    REG4, REG3           9243
```

Result of STBR Instruction

(REG4) = 13577531
(REG3) = 24688631
Condition Code = Unchanged

Compare Logical (CL)
Compare Logical Register (CLR)
Compare Logical Immediate (CLI)

Assembler Notation		Op-Code	Format
CL	R1,D2 (X2)	55	RX1,RX2
CL	R1,A2 (FX2,SX2)	55	RX3
CLR	R1,R2	05	RR
CLI	R1,I2 (X2)	F5	RI2

Operation

The first operand, the contents of the register specified by R1, is compared logically to the second operand. The result is indicated by the Condition Code setting. Neither operand is changed.

Condition Code

C	V	G	L	
0	X	0	0	First operand equal to second
1	X	0	1	First operand less than second
1	X	1	0	First operand less than second
0	X	0	1	First operand greater than second
0	X	1	0	First operand greater than second

Programming Note

In the RX formats, the second operand must be located on a fullword boundary.

The state of the V flag is undefined.

It is meaningful to check the following condition code mask (M1) after a logical comparison:

Mask	True/False*	Inference
3	False	First operand equal to second
3	True	First operand not equal to second
8	False	First operand greater than second
8	True	First operand less than second

*Refer to page 3-1 for True/False concept in branch instructions.

INSTRUCTIONS

Compare Logical Halfword (CLH)
Compare Logical Halfword Immediate (CLHI)

Assembler Notation		Op-Code	Format
CLH	R1,D2 (X2)	45	RX1,RX2
CLH	R1,A2 (FX2,SX2)	45	RX3
CLHI	R1,I2 (X2)	C5	RI1

Operation

The halfword second operand is expanded to a fullword by propagating the most significant bit through Bits 15:0. The first operand, the contents of the register specified by R1, is compared to this fullword. The result is indicated by the Condition Code setting. Neither operand is changed.

Condition Code

C	V	G	L	
0	X	0	0	First operand equal to second
1	X	0	1	First operand less than second
1	X	1	0	First operand less than second
0	X	0	1	First operand greater than second
0	X	1	0	First operand greater than second

Programming Note

In the RX formats, the second operand must be located on a halfword boundary.

The state of the V flag is undefined.

It is meaningful to check the following condition code mask (M1) after a logical comparison:

Mask	True/False*	Inference
3	False	First operand equal to second
3	True	First operand not equal to second
8	False	First operand greater than second
8	True	First operand less than second

*Refer to page 3-1 for True/False concept in branch instructions.

INSTRUCTION

Compare Logical Byte (CLB)

Assembler Notation		Op-Code	Format
CLB	R1,D2 (X2)	D4	RX1,RX2
CLB	R1,A2 (FX2,SX2)	D4	RX3

Operation

The byte quantity, contained in Bits 24:31 of the register specified by R1, is compared with the 8-bit second operand. The result is indicated by the Condition Code setting. Neither operand is changed.

Condition Code

C	V	G	L	
0	X	0	0	First operand equal to second
1	X	0	1	First operand less than second
1	X	1	0	First operand less than second
0	X	0	1	First operand greater than second
0	X	1	0	First operand greater than second

Programming Note

It is meaningful to check the following condition code mask (M1) after a logical comparison:

Mask	True/False*	Inference
3	False	First operand equal to second
3	True	First operand not equal to second
8	False	First operand greater than second
8	True	First operand less than second

*Refer to page 3-1 for True/False concept in branch instructions.

INSTRUCTIONS

AND (N)
AND Register (NR)
AND Immediate (NI)

Assembler Notation		Op-Code	Format
N	R1, D2 (X2)	54	RX1, RX2
N	R1, A2 (FX2, SX2)	54	RX3
NR	R1, R2	04	RR
NI	R1, I2 (X2)	F4	RI2

Operation

The logical product of the 32 bit second operand and the contents of the register specified by R1 replace the contents of the register specified by R1. The 32 logical bit product is formed on a bit-by-bit basis.

Condition Code

C	V	G	L	
0	0	0	0	Result is ZERO
0	0	0	1	Result is not ZERO
0	0	1	0	Result is not ZERO

Programming Note

In the RX formats, the second operand must be located on a fullword boundary.

When operating on fixed-point data, the Condition Code indicates ZERO (no flags), negative (L flag) or positive (G flag) result.

AND Halfword (NH)
AND Halfword Immediate (NHI)

Assembler Notation		Op-Code	Format
NH	R1,D2 (X2)	44	RX1,RX2
NH	R1,A2 (FX2,SX2)	44	RX3
NHI	R1,I2 (X2)	C4	RI1

Operation

The halfword second operand is expanded to a fullword by propagating the most significant bit through Bits 15:0. The logical product of this 32 bit quantity and the contents of the register specified by R1 replace the contents of the register specified by R1. The 32 bit logical product is formed on a bit-by-bit basis.

Condition Code

C	V	G	L	
0	0	0	0	Result is ZERO
0	0	0	1	Result is not ZERO
0	0	1	0	Result is not ZERO

Programming Note

In the RX formats, the second operand must be located on a halfword boundary.

When operating on fixed-point data, the Condition Code indicates ZERO (no flags), negative (Lflag) or positive (G flag) result.

OR (O)
OR Register (OR)
OR Immediate (OI)

Assembler Notation		Op-Code	Format
O	R1, D2 (X2)	56	RX1, RX2
O	R1, A2 (FX2, SX2)	56	RX3
OR	R1, R2	06	RR
OI	R1, I2 (X2)	F6	RI2

Operation

The logical sum of the 32 bit second operand and the contents of the register specified by R1 replace the contents of the register specified by R1. The logical sum is formed on a bit-by-bit basis.

Condition Code

C	V	G	L	
0	0	0	0	Result is ZERO
0	0	0	1	Result is not ZERO
0	0	1	0	Result is not ZERO

Programming Note

In the RX formats, the second operand must be located on a fullword boundary.
When operating on fixed-point data, the Condition Code indicates ZERO (no flags), negative (L flag) or positive (G flag) result.

OR Halfword (OH)
OR Halfword Immediate (OHI)

Assembler Notation		Op-Code	Format
OH	R1, D2 (X2)	46	RX1, RX2
OH	R1, A2 (FX2, SX2)	46	RX3
OHI	R1, I2 (X2)	C6	RI1

Operation

The halfword second operand is expanded to a fullword by propagating the most significant bit through Bits 15:0. The logical sum of this 32 bit quantity and the contents of the register specified by R1 replace the contents of the register specified by R1. The 32 bit sum is formed on a bit-by-bit basis.

Condition Code

C	V	G	L	
0	0	0	0	Result is ZERO
0	0	0	1	Result is not ZERO
0	0	1	0	Result is not ZERO

Programming Note

In the RX formats, the second operand must be located on a halfword boundary.

When operating on fixed-point data, the Condition Code indicates ZERO (no flags), negative (L flag) or positive (G flag) result.

INSTRUCTIONS

Exclusive OR (X)
Exclusive OR Register (XR)
Exclusive OR Immediate (XI)

Assembler Notation		Op-Code	Format
X	R1, D2 (X2)	57	RX1, RX2
X	R1, A2 (FX2, SX2)	57	RX3
XR	R1, R2	07	RR
XI	R1, I2 (X2)	F7	RI2

Operation

The logical difference of the 32 bit second operand and the contents of the register specified by R1 replace the contents of the register specified by R1. The 32 bit difference is formed on a bit-by-bit basis.

Condition Code

C	V	G	L	
0	0	0	0	Result is ZERO
0	0	0	1	Result is not ZERO
0	0	1	0	Result is not ZERO

Programming Note

In the RX formats, the second operand must be located on a fullword boundary.

When operating on fixed-point data, the Condition Code indicates ZERO (no flags), negative (L flag) or positive (G flag) result.

Exclusive OR Halfword (XH)
Exclusive OR Halfword Immediate (XHI)

Assembler Notation		Op-Code	Format
XH	R1, D2 (X2)	47	RX1, RX2
XH	R1, A2 (FX2, SX2)	47	RX3
XHI	R1, I2 (X2)	C7	RI1

Operation

The halfword second operand is expanded to a fullword by propagating the most significant bit through Bits 15:0. The logical difference of this 32 bit quantity and the contents of the register specified by R1 replace the contents of the register specified by R1. The 32 bit difference is formed on a bit-by-bit basis.

Condition Code

C	V	G	L	
0	0	0	0	Result is ZERO
0	0	0	1	Result is not ZERO
0	0	1	0	Result is not ZERO

Programming Note

In the RX formats, the second operand must be located on a halfword boundary.

When operating on fixed-point data, the Condition Code indicates ZERO (no flags), negative (L flag) or positive (G flag) result.

INSTRUCTION

Test Immediate (TI)

Assembler Notation		Op-Code	Format
TI	R1,I2 (X2)	F3	RI2

Operation

Each bit of the second operand is logically ANDed with the corresponding bit in the register specified by R1. Neither operand is changed.

Condition Code

C	V	G	L	
0	0	0	0	Result is ZERO
0	0	0	1	Result is not ZERO (Most significant bit is set)
0	0	1	0	Result is not ZERO (Most significant bit is reset)

Example: TI

Assembler Notation		Machine Code		Comments
LI	REG2, Y'62314020'	F820	62314020	(REG2) = 62314020
TI	REG2, Y'08000000'	F320	08000000	TEST IF BIT 4 IN REG2 IS SET
BP	X'1000'	4220	1000	DO NOT BRANCH AS CONDITION CODE IS 0000

INSTRUCTION

Test Halfword Immediate (THI)

Assembler Notation		Op-Code	Format
THI	R1,I2 (X2)	C3	RI1

Operation

The halfword second operand is expanded to a fullword by propagating the most significant bit through Bits 15:0. Each bit in this quantity is logically ANDed with the corresponding bit contained in the register specified by R1. Neither operand is changed.

Condition Code

C	V	G	L	
0	0	0	0	Result is ZERO
0	0	0	1	Result is not ZERO (Most significant bit is set)
0	0	1	0	Result is not ZERO (Most significant bit is reset)

Programming Note

When operating on fixed-point data, the Condition Code indicates ZERO (no flags), negative (L flag) or positive (G flag) result.

Shift Left Logical (SLL)
Shift Left Logical Short (SLLS)

Assembler Notation		Op-Code	Format
SLL	R1, I2 (X2)	ED	RI1
SLLS	R1, N	11	SF

Operation

The first operand, the contents of the register specified by R1, is shifted left the number of places specified by the second operand. Bits shifted out of Position 0 are shifted through the carry flag of the Condition Code and then lost. The last bit shifted remains in the carry flag. Zeros are shifted into Position 31.

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is not ZERO
X	0	1	0	Result is not ZERO
1	0	X	X	Carry

Programming Note

In the RI formats, the shift count is specified by the least significant five bits of the second operand.

In the SF format, the maximum shift count is 15.

The state of the C flag indicates the state of the last bit shifted out of Position 0.

If the second operand specifies a shift of zero places, the Condition Code is set in accordance with the value contained in the register. The state of the C flag is undefined in this case.

When the register specified by Rl contains fixed point data, the L flag set indicates a negative result, the G flag set indicates a positive result.

Shift Right Logical (SRL)
Shift Right Logical Short (SRLS)

Assembler Notation		Op-Code	Format
SRL	R1, I2 (X2)	EC	RI1
SRLS	R1, N	10	SF

Operation

The first operand, the contents of the register specified by R1, is shifted right the number of places specified by the second operand. Bits shifted out of Position 31 are shifted through the carry flag of the Condition Code and then lost. The last bit shifted remains in the carry flag. Zeros are shifted into Position 0.

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is not ZERO
X	0	1	0	Result is not ZERO
1	0	X	X	Carry

Programming Note

In the RI1 format, the shift count is specified by the least significant five bits of the second operand.

In the SF format, the maximum shift count is 15.

The state of the C flag indicates the state of the last bit shifted out of Position 31.

When the register specified by R1 contains fixed point data, the L flag set indicates a negative result, the G flag set indicates a positive result.

If the second operand specifies a shift of zero places, the Condition Code is set in accordance with the value contained in the register. The state of the C flag is undefined in this case.

Shift Left Halfword Logical (SLHL)
Shift Left Halfword Logical Short (SLHLS)

Assembler Notation		Op-Code	Format
SLHL	R1,I2 (X2)	CD	RI1
SLHLS	R1,N	91	SF

Operation

Bits 16:31 of the register specified by R1 are shifted left the number of places specified by the second operand. Bits shifted out of Position 16 are shifted through the carry flag and lost. The last bit shifted remains in the carry flag. Zeros are shifted into Position 31. Bits 0:15 of the first operand remain unchanged.

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is not ZERO
X	0	1	0	Result is not ZERO
1	0	X	X	Carry

Programming Note

The condition code setting is based on the halfword (bits 16:31) result.

In the RI1 format, the shift count is specified by the least significant four bits of the second operand.

In the SF format, the maximum shift count is 15.

The state of the C flag indicates the state of the last bit shifted out of Position 16.

When the register specified by R1 contains fixed point data, the L flag set indicates a negative result, the G flag set indicates a positive result.

If the second operand specifies a shift of zero places, the condition code is set in accordance with the value contained in bits 16:31 of the register. The state of the C flag is undefined, in this case.

Shift Right Halfword Logical (SRHL)
Shift Right Halfword Logical Short (SRHLS)

Assembler Notation		Op-Code	Format
SRHL	R1, I2 (X2)	CC	RI1
SRHLS	R1, N	90	SF

Operation

Bits 16:31 of the register specified by R1 are shifted right the number of places specified by the second operand. Bits shifted out of Position 31 are shifted through the carry flag and lost. The last bit shifted remains in the carry flag. Zeros are shifted into Position 16. Bits 0:15 of the first operand remain unchanged.

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is not ZERO
X	0	1	0	Result is not ZERO
1	0	X	X	Carry

Programming Note

The condition code setting is based on the halfword (bits 16:31) result.

In the RI1 format, the shift count is specified by the least significant four bits of the second operand.

In the SF format, the maximum shift count is 15.

The state of the C flag indicates the state of the last bit shifted out of the Position 31.

When the register specified by R1 contains fixed point data, the L flag set indicates a negative result, the G flag set indicates a positive result.

If the second operand specifies a shift of zero places, the Condition Code is set in accordance with the halfword value contained in bits 16:31 of the register. The state of the C flag is undefined in this case.

INSTRUCTION

Rotate Left Logical (RLL)

Assembler Notation		Op-Code	Format
RLL	R1, I2 (X2)	EB	RI1

Operation

The 32 bit first operand, contained in the register specified by R1, is shifted left, end around, the number of positions specified by the second operand. Bits shifted out of Position 0 are shifted into Position 31.

Condition Code

C	V	G	L	
0	0	0	0	Result is ZERO
0	0	0	1	Result is not ZERO
0	0	1	0	Result is not ZERO

Programming Note

The shift count is specified by the least significant five bits of the second operand.

When the register specified by R1 contains fixed point data, the L flag set indicates a negative result, the G flag set indicates a positive result.

If the second operand specifies a shift of zero places, the Condition Code is set in accordance with the value contained in the register specified by R1.

Example: RLL

1.

Assembler Notation	Machine Code	Comments
LI REG9, Y'56789ABC'	F890 56789ABC	(REG 9) = 56789ABC
RLL REG9, X'0004'	EB90 0004	

Result of RLL Instruction

(REG 9) = 6789ABC5
Condition Code = 0010 (G = 1)

2.

Assembler Notation	Machine Code	Comments
LI REG9, Y'88880000'	F890 8888 0000	(REG 9) = 88880000
RLL REG9, X'03'	EB90 0003	

Result of RLL Instruction

(REG 9) = 44400004
Condition Code = 0010 (G = 1)

INSTRUCTION

Rotate Right Logical (RRL)

Assembler Notation		Op-Code	Format
RRL	R1, I2 (X2)	EA	RI1

Operation

The 32 bit first operand, contained in the register specified by R1, is shifted right, end around, the number of positions specified by the second operand. Bits shifted out of Position 31 are shifted into Position 0.

Condition Code

C	V	G	L	
0	0	0	0	Result is ZERO
0	0	0	1	Result is not ZERO
0	0	1	0	Result is not ZERO

Programming Note

The shift count is specified by the least significant five bits of the second operand.

When the register specified by R1 contains fixed point data, the L flag set indicates a negative result, the G flag set indicates a positive result.

If the second operand specifies a shift of zero places, the Condition Code is set in accordance with the value contained in the register specified by R1.

Example: RRL

1.

Assembler Notation		Machine Code	Comments
LI	REG4, Y'12345678'	F840 1234 5678	(REG4) = 12345678
RLL	REG4, X'04'	EA40 0004	

Result of RRL Instruction

(REG4) = 81234567
Condition Code = 0001 (L = 1)

2.

Assembler Notation		Machine Code	Comments
LI	REG4, Y'00001111'	F840 0000 1111	(REG 4) = 00001111
RRL	REG4, X'01'	EA40 0001	

Result of RRL Operation

(REG4) = '800000888'
Condition Code = 0001 (L = 1)

INSTRUCTION

Test and Set (TS)

Assembler Notation		Op-Code	Format
TS	D2 (X2)	E0	RX1, RX2
TS	A2 (FX2, SX2)	E0	RX3

Operation

The halfword second operand is read from memory and, on the same cycle, written back with the most significant bit set. The most significant bit of the second operand is tested. The Condition Code reflects the state of this bit at the time of the memory read. The other bits in the halfword are undefined.

Condition Code

C	V	G	L	
X	X	X	0	Most significant bit reset
X	X	X	1	Most significant bit set

Programming Note

The Test and Set instruction provides a mechanism for software synchronization.

The Test and Set instruction can be used in a single processor environment as follows. Two or more user tasks running under an Operating System share a halfword. This halfword is located in a memory area referred to as Task Common. Each task can access the halfword using the TS instruction. The synchronization sequence may be as follows:

TASK 1 Sets the most significant bit using the TS instruction.

TASK 2 Senses the most significant bit using the TS instruction, sees that it is set, performs the necessary software synchronization, and then resets the most significant bit of the halfword.

The Test and Set instruction can be used in a multi-processor system as follows. Two or more processors share a halfword. This halfword is located in a memory area can access the halfword using the TS instruction. The synchronization sequence can be exactly as explained for user tasks with the following subtle difference. Whereas, TASK 1 and TASK 2 cannot access the halfword at the same (real) time, two processors can. The access is granted according to the priority.

The hardware/firmware insures that no other accesses to the halfword have been made during the execution of the TS instruction.

INSTRUCTION

Test Bit (TBT)

Assembler Notation		Op-Code	Format
TBT	R1,D2 (X2)	74	RX1,RX2
TBT	R1,A2 (FX2,SX2)	74	RX3

Operation

The second operand address points to a bit array starting on a halfword boundary. The value contained in the register specified by R1 is the bit displacement into the array. The bit is located and tested. The test does not change the bit.

Condition Code

C	V	G	L	
0	0	0	0	Tested bit is ZERO
0	0	1	0	Tested bit is ONE

Example: TBT

Assembler Notation		Machine Code	Comments
LIS	REG8, 3	2483	(REG 8) = 3
TBT	REG8, LABEL	7480 0BC4	LABEL = Halfword in memory = X'B34A'

Result of TBT Instruction

Memory Location X'BC4' unchanged
(REG 8) unchanged
Condition Code = 0010 (G = 1)

INSTRUCTION

Set Bit (SBT)

Assembler Notation		Op-Code	Format
SBT	R1, D2 (X2)	75	RX1, RX2
SBT	R1, A2 (FX2, SX2)	75	RX3

Operation

The second operand address points to a bit array starting on a halfword boundary. The value contained in the register specified by R1 is the bit displacement into the array. The bit is located and forced to one.

Condition Code

C	V	G	L	
0	0	0	0	Previous state of bit was ZERO
0	0	1	0	Previous state of bit was ONE

Example: SBT

Assembler Notation		Machine Code	Comments
LIS	REG5, 8	2458	(REG 5) = 8
SBT	REG5, LABEL	7550 1520	LABEL Located at X'1520'. It contains X'2134'.

Result of SBT Instruction

Contents of LABEL = 21B4
(REG 5) unchanged
Condition Code = 0000 (G = 0)

INSTRUCTION

Complement Bit (CBT)

Assembler Notation		Op-Code	Format
CBT	R1,D2 (X2)	77	RX1,RX2
CBT	R1,A2 (FX2,SX2)	77	RX3

Operation

The second operand address points to a bit array starting on a halfword boundary. The value contained in the register specified by R1 is the bit displacement into the array. The bit is located and complemented.

Condition Code

C	V	G	L	
0	0	0	0	Previous state of bit was ZERO
0	0	1	0	Previous state of bit was ONE

Example: CBT

Assembler Notation		Machine Code	Comments
LIS	REG9,3	2493	(REG 9) = 3
CBT	REG9, LABEL	7790 0C4A	LABEL located at X'C4A'. It contains X'2813'.

Result of CBT Instruction

Contents of LABEL = 3813
(REG9) unchanged
Condition Code = 0000 (G = 0)

INSTRUCTION

Reset Bit (RBT)

Assembler Notation		Op-Code	Format
RBT	R1,D2 (X2)	76	RX1,RX2
RBT	R1,A2 (FX2,SX2)	76	RX3

Operation

The second operand address points to a bit array starting on a halfword boundary. The value contained in the register specified by R1 is the bit displacement into the array. The bit is located and forced to ZERO.

Condition Code

C	V	G	L	
0	0	0	0	Previous state of bit was ZERO
0	0	1	0	Previous state of bit was ONE

Example: RBT

Assembler Notation		Machine Code	Comments
LIS	REG2,3	2423	(REG 2) = 3
RBT	REG2,LABEL	7620 1A42	LABEL located at X'1A42' contains X'3143'

Result of RBT Instruction

Contents of LABEL = 2143
(REG 2) unchanged
Condition Code = 0010 (G = 1)

INSTRUCTIONS

Cyclic Reduncancy Check Modulo 12 (CRC12)
Cyclic Redundancy Check Modulo 16 (CRC16)

Assembler Notation		Op-Code	Format
CRC12	R1,D2 (X2)	5E	RX1,RX2
CRC12	R1,A2 (FX2,SX2)	5E	RX3
CRC16	R1,D2 (X2)	5F	RX1,RX2
CRC16	R1,A2 (FX2,SX2)	5F	RX3

Operation

These instructions are used to generate either a 12 bit or a 16 bit Cyclic Redundancy Check (CRC) character. The register specified by R1 contains, in Bits 24:31, the next data character to be included in the CRC. The second operand is the accumulated (old) CRC. The polynominal used for the 12 bit CRC generation is:

$$X^{12}+ X^{11}+ X^{3}+ X^{2}+ X + 1$$

The polynomial used for the 16 bit CRC generation is:

$$X^{16}+ X^{15}+ X^{2}+ 1$$

The second operand is replaced by the generated CRC character.

Condition Code

Unchanged

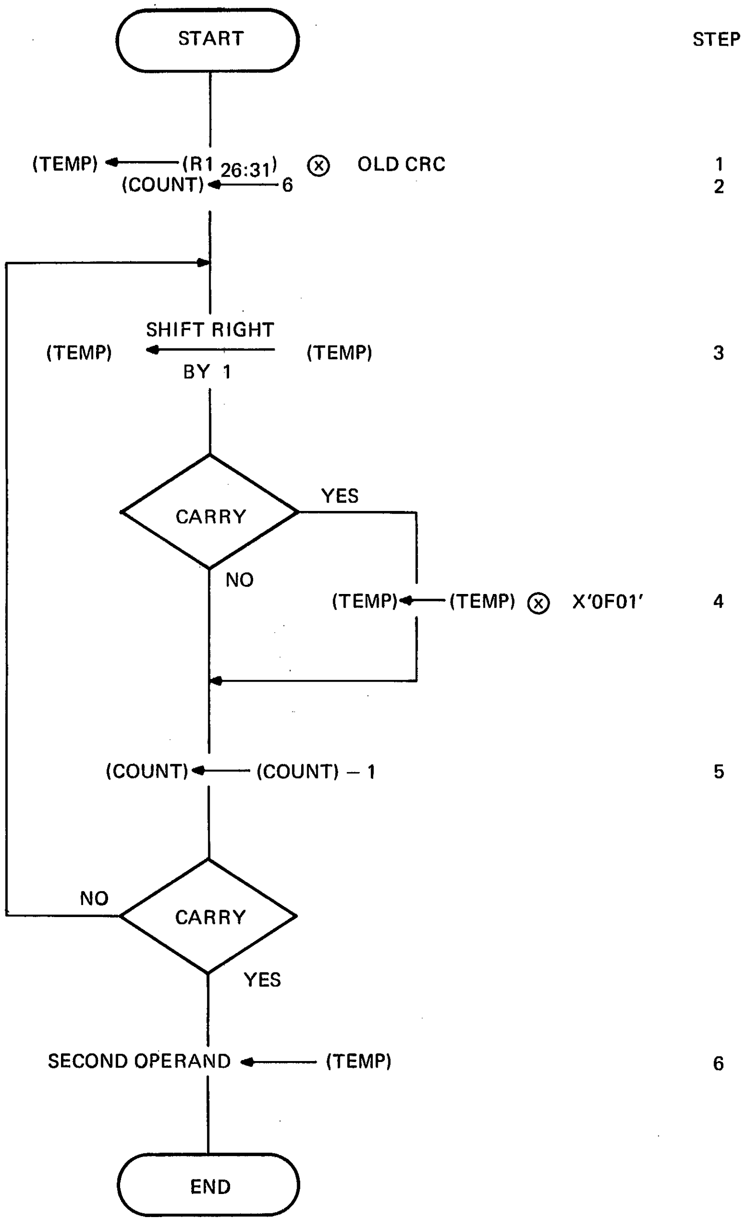
Programming Note

The register specified by Rl remains unchanged.

The second operand must be located on a halfword boundary.

Figure 2-5 illustrates a Flow Chart for CRC generation.

CRC12 ALGORITHM



FOR CRC 16 ALGORITHM, USE:	$R1_{23:31}$	INSTEAD OF $R1_{26:31}$	IN STEP 1
	8	INSTEAD OF 6	IN STEP 2
	X'A001'	INSTEAD OF X'0F01'	IN STEP 4

Figure 2-5. Flow Chart for CRC Generation

INSTRUCTION

Translate (TLATE)

Assembler Notation		Op-Code	Format
TLATE	R1,D2 (X2)	E7	RX1,RX2
TLATE	R1,A2 (FX2,SX2)	E7	RX3

Operation

The least significant bits (Bits 24:31) of the register specified by R1 contain the character to be translated. The fullword location specified by the second operand address contains the address of a translation table. The table is made up of 256 halfwords. The character contained in the register specified by R1 is used as an index into the table.

If Bit 0 of the table entry corresponding to the index character is one, then Bits 8:15 of the table entry replace the index character, and the next sequential instruction is executed.

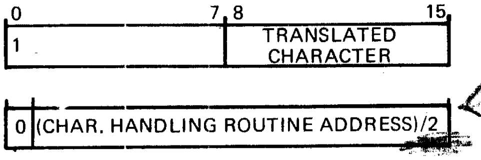
If Bit 0 of the table entry is zero, then Bits 1:15 of the table entry contain the address, divided by two, of a special handling routine. In this case, no translation takes place. The address contained in Bits 1:15 is shifted left by one, (multiplied by two). This address replaces the current Location Counter, thereby effecting an unconditional branch.

Condition Code

Unchanged

Programming Note

The second operand address must be aligned on a fullword boundary.



Example: TLATE

This example illustrates the use of the TLATE instruction. The translation table must either be initialized or assembled to contain up to a total of 256 halfword entries. In this example, the table contains 2 entries:

```
Label           Assembler Notation          Comments

                LHI   REG5,X'8Ø52'          LOAD TABLE ENTRY INTO REG5
                STH   REG5,TABLE            PUT ENTRY INTO TABLE
                LA    REG7,TRANLAB          LOAD ANOTHER TABLE ENTRY
                SRLS  REG7,1                DIVIDE BY 2
                STH   REG7,TABLE+A          PUT ENTRY INTO TABLE
                .
                .
                .
TABADR          DC    A(TABLE)
```

Alternately, this table may be assembled with the proper constant values. The T type constant may be used to assemble subroutine addresses in the proper format. For example:

```
        ALIGN   2
TABLE   EQU     *
        DO      256
        DC      H'0'
        ORG     TABLE + 4
        DC      T(TRANLAB)
        ORG     TABLE + 512
```

Since a program is normally assembled as a relocatable program, the Address of TRANLAB is not known, but for illustrative purposes assume address of TRANLAB is X'864'.

	0 15
TABLE+0	
TABLE+2	
TABLE+4	8052
TABLE+6	
TABLE+8	
TABLE+A	0432
TABLE+C	
TABLE+508	

At TABLE+A is the address of TRANLAB divided by 2 (X'864'/2)

1. Using this table, this example translates the character in Register 2.

```
Label              Assembler Notation            Comments

                   LIS    REG2,2                 (REG 2) = 0000 0002
                   TLATE    REG2, TABADR
```

Result of TLATE Instruction

(REG2) = 0000 0052
Condition Code = Unchanged

The entry used = Contents at Address of (2 times contents of REG 2) + TABLE
= Contents at address TABLE + 4
= X'8052'

Since first bit of entry = 1, Direct translation is used and the contents of REG2 are replaced by X'0000 0052'

2. Using the table, the following example shows how the TLATE instruction can be used to branch to a special character handling routine:

```
Label              Assembler Notation            Comments

                   LIS    REG5,5                 REG5 = 0000 0005
                   TLATE    REG5, TABADR
                   •
                   •
                   •
                   •
                   •
TRANLAB            LHR    R6,R5                  THESE INSTRUCTIONS
                   LB    R3,0 (R6)               OPERATE ON THE SPECIAL
                                                 CHARACTER.
                   •
                   •
                   •

                   •
                   •
                   •
```

Result of TLATE Instruction

(REG5) = 0000 0005
Condition Code = Unchanged

Control is Transferred to subroutine at address TRANLAB (X'864').

The entry used = Contents at Address of (2 times contents of REG 5) + TABLE
= Contents at Address TABLE + A
= X'0432'

Since the first bit of entry = 0, the microcode multiplies the entry by 2 and transfers to TRANLAB (at address X'864') and continues executing instructions from the new address.

INSTRUCTIONS

Add to Top of List (ATL)
Add to Bottom of List (ABL)

Assembler Notation		Op-Code	Format
ATL	R1,D2 (X2)	64	RX1,RX2
ATL	R1,A2 (FX2,SX2)	64	RX3
ABL	R1,D2 (X2)	65	RX1,RX2
ABL	R1,A2 (FX2,SX2)	65	RX3

Operation

The register specified by R1 contains the fullword element to be added to the list. The list is located in memory at the address of the second operand. The number of slots used tally is compared with the number of slots in the list. If the number of slots used equals the number of slots in the list, an overflow condition exists. The element is not added to the list and the overflow flag in the Condition Code is set. If the number of slots used tally is less than the number of slots in the list, it is incremented by one, the appropriate pointer is changed, and the element is added to the list. Refer to Figure 2-4.

Condition Code

C	V	G	L	
0	0	0	0	Element added successfully
0	1	0	0	List overflow

Programming Note

These instructions manipulate circular lists as described in the introduction to this chapter.

The second operand location must be on a fullword boundary.

The add to top of list instruction manipulates the current top pointer in the list. If no overflow occurs, the current top pointer, which points to the last element added to the top of the list, is decremented by one and the element is inserted in the slot pointed to by the new current top pointer. If the current top pointer was zero on entering this instruction, the current top pointer is set to the maximum slot number in the list. This condition is referred to as list wrap.

The add to bottom of list instruction manipulates the next bottom pointer. If no overflow occurs, the element is inserted in the slot pointed to by the next bottom pointer, and the next bottom pointer is incremented by one. If the incremented next bottom pointer is greater than the maximum slot number in the list, the next bottom pointer is set to zero. This condition is referred to as list wrap.

See examples in the next section.

INSTRUCTIONS

Remove from Top of List (RTL)
Remove from Bottom of List (RBL)

Assembler Notation		Op-Code	Format
RTL	R1,D2 (X2)	66	RX1,RX2
RTL	R1,A2 (FX2,SX2)	66	RX3
RBL	R1,D2 (X2)	67	RX1,RX2
RBL	R1,A2 (FX2,SX2)	67	RX3

Operation

The element removed from the list replaces the contents of the register specified by R1. The list is located at the address of the second operand. If, at the start of the instruction execution, the number of slots used tally is ZERO, the list is already empty and the instruction terminates with the overflow flag set in the Condition Code. This condition is referred to as list underflow; in this case, R1 is undefined. If underflow does not occur, the number of slots used tally is decremented by one, the appropriate pointer is changed, and the element is extracted and placed in the register specified by R1.

Condition Code

C	V	G	L	
0	0	0	0	List now empty
0	0	1	0	List is not yet empty
0	1	0	0	List was already empty, dummy

Programming Note

These instructions manipulate circular lists as described in the introduction to this chapter.

The second operand location must be on a fullword boundary.

In the case of list underflow, the contents of the register specified by R1 are undefined.

The remove from top of list instruction manipulates the current top pointer. If no underflow occurs, the current top pointer points to the element to be extracted. The element is extracted, and placed in the register specified by R1. The current top pointer is incremented by one and compared to the maximum slot number. If the current top pointer is greater than the maximum slot number, the current top pointer is set to ZERO. This condition is referred to as list wrap.

The remove from bottom of list instruction manipulates the next bottom pointer. If no underflow occurs, and the next bottom pointer is ZERO, it is set to the maximum slot number (list wrap); otherwise, it is decremented by one, and the element now pointed to is extracted and placed in the register specified by R1.

Examples: List Instructions (ATL, ABL, RTL, RBL)

The following are examples of the use of the four list processing instructions.

The original list is normally set up as shown in Figure 2-6.

LIST	0005	0000
	0000	0000
SLOT 0	UNDEFINED	
SLOT 1	UNDEFINED	
SLOT 2	UNDEFINED	
SLOT 3	UNDEFINED	
SLOT 4	UNDEFINED	

where HALFWORDS at

LIST	=	# of total slots
	=	5 (in this example)
LIST + 2	=	# of entries used
	=	0
LIST + 4	=	current top of list
	=	slot 0
LIST + 6	=	next bottom of list
	=	slot 0

Figure 2-6. List Processing Instructions

Assembler Notation		Results and Comments
LIS	REG0,0	
STH	REG0,LIST+2	INITIALIZE # OF ENTRIES USED TO 0
ST	REG0,LIST+4	INITIALIZE POINTERS TO 0
LIS	REG1,1	REGISTERS 1 THRU 6 CONTAIN
LIS	REG2,2	1 THRU 6 RESPECTIVELY
LIS	REG3,3	
LIS	REG4,4	
LIS	REG5,5	
LIS	REG6,6	
STH	REG5,LIST	TOTAL # OF ENTRIES = 5

REF1 ATL REG1, LIST

LIST	0005	0001	
	0004	0000	(List Wrap)
SLOT 0	UNDEFINED		
SLOT 1	UNDEFINED		
SLOT 2	UNDEFINED		
SLOT 3	UNDEFINED		
SLOT 4	0000 0001		

Condition Code = 0000
Current Top Pointer = Slot 4
Next Bottom Pointer = Slot 0

REF2 ATL REG2, LIST

LIST	0005	0002
	0003	0000
SLOT 0	UNDEFINED	
SLOT 1	UNDEFINED	
SLOT 2	UNDEFINED	
SLOT 3	0000 0002	
SLOT 4	0000 0001	

Condition Code = 0000
Current Top Pointer = Slot 3
Next Bottom Pointer = Slot 0

REF3 ATL REG3, LIST

LIST	0005	0003
	0002	0000
SLOT 0	UNDEFINED	
SLOT 1	UNDEFINED	
SLOT 2	0000 0003	
SLOT 3	0000 0002	
SLOT 4	0000 0001	

Condition Code = 0000
Current Top Pointer = Slot 2
Next Bottom Pointer = Slot 0

REF4 ABL REG4, LIST

LIST	0005	0004
	0002	0001
SLOT 0	0000 0004	
SLOT 1	UNDEFINED	
SLOT 2	0000 0003	
SLOT 3	0000 0002	
SLOT 4	0000 0001	

Condition Code = 0000
Current Top Pointer = Slot 2
Next Bottom Pointer = Slot 1

REF5 ABL REG5, LIST

LIST	0005	0005
	0002	0002
SLOT 0	0000 0004	
SLOT 1	0000 0005	
SLOT 2	0000 0003	
SLOT 3	0000 0002	
SLOT 4	0000 0001	

Condition Code = 0000
Current Top Pointer = Slot 2
Next Bottom Pointer = Slot 2

REF6 ABL REG6, LIST

LIST	0005	0005
	0002	0002
SLOT 0	0000 0004	
SLOT 1	0000 0005	
SLOT 2	0000 0003	
SLOT 3	0000 0002	
SLOT 4	0000 0001	

Condition Code = 0100 (List overflow)
Current Top Pointer = Slot 2
Next Bottom Pointer = Slot 2

REF7 RTL REG7, LIST

LIST		0005	0004
		0003	0002
SLOT 0		0000 0004	
SLOT 1		0000 0005	
SLOT 2	X	0000 0003	
SLOT 3		0000 0002	
SLOT 4		0000 0001	

(REG 7) = 0000 0003
Condition Code = 0010
Current Top Pointer = Slot 3
Next Bottom Pointer = Slot 2

REF8 RBL REG8, LIST

LIST		0005	0003
		0003	0001
SLOT 0		0000 0004	
SLOT 1	X	0000 0005	
SLOT 2	X	0000 0003	
SLOT 3		0000 0002	
SLOT 4		0000 0001	

(REG) 8) = 0000 0005
Condition Code = 0010
Current Top Pointer = Slot 3
Next Bottom Pointer = Slot 1

REF9 RTL REG9, LIST

LIST		0005	0002
		0004	0001
SLOT 0		0000 0004	
SLOT 1	X	0000 0005	
SLOT 2	X	0000 0003	
SLOT 3	X	0000 0002	
SLOT 4		0000 0001	

(REG9) = 0000 0002
Condition Code = 0010
Current Top Pointer = Slot 4
Next Bottom Pointer = Slot 1

NOTE

X = Entry removed from list, and is not accessible through further manipulation of list instructions.

REF10 RBL REG10, LIST

LIST		0005	0001
		0004	0000
SLOT 0	X	0000	0004
SLOT 1	X	0000	0005
SLOT 2	X	0000	0003
SLOT 3	X	0000	0002
SLOT 4		0000	0001

(REG 10) = 0000 0004
Condition Code = 0010
Current Top Pointer = Slot 4
Next Bottom Pointer = Slot 0

REF11 RTL REG11, LIST

LIST		0005	0000
		0000	0000
SLOT 0	X	0000	0004
SLOT 1	X	0000	0005
SLOT 2	X	0000	0003
SLOT 3	X	0000	0002
SLOT 4	X	0000	0001

(REG 11) = 0000 0001
Condition Code = 0000 (List is now empty)
Current Top Pointer = Slot 0
Next Bottom Pointer = Slot 0

REF12 RTL REG12, LIST

LIST		0005	0000
		0000	0000
SLOT 0	X	0000	0004
SLOT 1	X	0000	0005
SLOT 2	X	0000	0003
SLOT 3	X	0000	0002
SLOT 4	X	0000	0001

(REG 12) = UNDEFINED
Condition Code = 0100 (List was already empty)
Current Top Pointer = Slot 0
Next Bottom Pointer = Slot 0

NOTE

X= Entry removed from list, and is not accessible through further manipulation of list instructions.

CHAPTER 3
BRANCHING

In normal operations, the Processor executes instructions in sequential order. The Branch instructions allow this sequential mode of operation to be varied, so that programs can loop, transfer control to subroutines, or make decisions based on the results of previous operations.

OPERATIONS

The second operand in Branch instructions is the address of the memory location to which control is transferred. The address may be contained in a register or it may be specified in the instruction as the second operand address.

Decision Making

The Conditional Branch instructions permit the program to make the decisions based on previous results. In these instructions, the R1 field contains a four bit mask, M1, which is tested against the Condition Code. The result of the test determines whether the branch is taken, or the next sequential instruction is executed.

The following examples show previous Condition Code, mask specified in a branch instruction, and the result of the test on which branch or no branch decision is made.

Previous Condition Code	Mask(M1)	Result of Test	(True/False)
0000	0010	0000	(False)
0001	1010	0000	(False)
1001	1000	1000	(True)
0100	0100	0100	(True)
1010	0010	0010	(True)
0010	0011	0010	(True)
0010	0000	0000	(False)

Subroutine Linkage

The Branch and Link instructions allow branching to subroutines in such a way that a return address is passed to the subroutine. In these instructions, the address of the instruction immediately following the Branch instruction is saved in the register specified by R1.

BRANCH INSTRUCTION FORMATS

The Branch instructions use the Register to Register (RR), the Short Form (SF), and the Register and Indexed Storage (RX) formats.

BRANCH INSTRUCTIONS

The instructions described in this section are:

BFC	Branch on False Condition
BFCR	Branch on False Condition Register
BFBS	Branch on False Condition Backward Short
BFFS	Branch on False Condition Forward Short
BTC	Branch on True Condition
BTCR	Branch on True Condition Register
BTBS	Branch on True Condition Backward Short
BTFS	Branch on True Condition Forward Short
BAL	Branch and Link
BALR	Branch and Link Register
BXLE	Branch on Index Low or Equal
BXH	Branch on Index High

INSTRUCTIONS

Branch on True Condition (BTC)
Branch on True Condition Register (BTCR)
Branch on True Condition Backward Short (BTBS)
Branch on True Condition Forward Short (BTFS)

Assembler Notation		Op-Code	Format
BTC	M1, D2 (X2)	42	RX1, RX2
BTC	M1, A2 (FX2, SX2)	42	RX3
BTCR	M1, R2	02	RR
BTBS	M1, N	20	SF
BTFS	M1, N	21	SF

Operation

The Condition Code of the Program Status Word is tested for the conditions specified by the mask field, M1. If any of the conditions tested are found to be true, a branch is executed to the second operand location. If none of the conditions tested is found to be true, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

In the RR format, the branch address is contained in the register specified by R2.

In the SF format, the N field contains the number of halfwords to be added or subtracted from the current Location Counter to obtain the branch address.

In the RR and RX formats, the branch address must be located on a halfword boundary.

Example: BTC

Assembler Notation		Machine Code	Comments
LH	R1, X'100'	4810 0100	Load halfword (X'1234') located at X'100' Condition Code is set to CVGL = 0010 Mask is 3, i.e., M1 = 0011. Perform logical AND between CVGL and M1, i.e., 0010 and 0011. The result is 0010, i.e., true; therefore, a branch is taken to LOC.
BTC	3, LOC	4230 ABC0	

INSTRUCTIONS

Branch on False Condition (BFC)
Branch on False Condition Register (BFCR)
Branch on False Condition Backward Short (BFBS)
Branch on False Condition Forward Short (BFFS)

Assembler Notation		Op-Code	Format
BFC	M1, D2 (X2)	43	RX1, RX2
BFC	M1, A2 (FX2, SX2)	43	RX3
BFCR	M1, R2	03	RR
BFBS	M1, N	22	SF
BFFS	M1, N	23	SF

Operation

The Condition Code of the Program Status Word is tested for the conditions specified in the mask field, M1. If all conditions tested are found to be false, a branch is executed to the second operand location. If any of the conditions tested is found to be true, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

In the RR format, the branch address is contained in the register specified by R2.

In the SF format, the N field contains the number of halfwords to be added to or subtracted from the current Location Counter to obtain the branch address.

In the RR and RX formats, the branch address must be located on a halfword boundary.

Example: BFC

Assembler Notation		Machine Code	Comments
LCS	R1, 2	2512	(R1) = FFFFFFFE. Condition Code, CVGL = 0001 Mask is 1001. Perform logical AND between mask and CVGL, i. e., 1001 and 0001. The result in 0001, i. e., true, therefore, a branch is not taken in LOC.
BFC	9, LOC	4390 ABC0	

INSTRUCTIONS

Branch and Link (BAL)
Branch and Link Register (BALR)

Assembler Notation		Op-Code	Format
BAL	R1,D2 (X2)	41	RX1,RX2
BAL	R1,A2 (FX2,SX2)	41	RX3
BALR	R1,R2	01	RR

Operation

The address of the next sequential instruction is saved in the register specified by R1, and a branch is taken to the second operand address.

Condition Code

Unchanged

Programming Note

The second operand location must be on a halfword boundary.

The branch address is calculated before the register specified by Rl is changed. Rl may specify the same register as X2, FX2, SX2, or R2.

Example: BAL

The following example illustrates the use of the BAL instruction. The instruction causes control to be transferred to a subroutine called SUBROUT. After completion of the subroutine, the linking register is used to branch back to the next sequential instruction after the BAL; i.e., the instruction labelled RETURN.

	Label	Assembler Notation	Comments
	BEGIN	BAL REG4, SUBROUT	TRANSFER TO SUBROUT
MAIN	RETURN	XR R6,R6	
PROG		STH R6,LAB+4	
		:	
		:	
	SUBROUT	LHL R8,LOC	THE RETURN ADDRESS OF THE SUBROUTINE IS IN REG4
SUBROUTINE		AHI R8,10	
		:	
		:	
	RTNEND	BR REG4	RETURN TO XR INST.

NOTE

Within the subroutine, the linking register (REG4 in the example) should not be used.

Result of BAL Instruction

Condition Code = Unchanged

INSTRUCTION

Branch on Index Low or Equal (BXLE)

Assembler Notation		Op-Code	Format
BXLE	R1,D2 (X2)	C1	RX1,RX2
BXLE	R1,A2 (FX2,SX2)	C1	RX3

Set Up

	0 31
R1	Starting index value
R1+1	Increment value
R1+2	Limit or final value

Prior to execution of this instruction, the register specified by R1 must contain a starting index value. The register specified by R1+1 must contain an increment value. The register specified by R1+2 must contain a comparand (limit or final value). All values may be signed.

Operation

Execution of this instruction causes the increment value to be added to the index value. The result is logically compared to the limit or final value. If the index value is less than or equal to the limit value, a branch is executed to the second operand location. If the index value is greater than the limit value, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The incremented index value replaces the contents of the register specified by R1.

The register numbers wrap around, i.e., three consecutive registers used by this instruction, may be 6, 7, 8 or 14, 15, 0 or 15, C, 1, etc.

The second operand location must be on a halfword boundary.

The branch address is calculated before incrementing the starting index value contained in the register specified by R1.

The register specified by R1 may be the same as X2, FX2 or SX2.

Example: BXLE

Transfer 10 bytes in memory starting at Memory Location Labelled BUF0 to memory location labelled BUF1.

```
Labels      Assembler Notation       Comments

            LIS    REG3,0            (REG 3) = STARTING INDEX VALUE = 0
            LIS    REG4,1            (REG 4) = INCREMENT VALUE
            LIS    R5,9              (REG 5) = FINAL VALUE = 9
AGAIN       LB     REG0, BUFO(R3)    (REG 0) = 1 BYTE FROM BUF0
            STB    REG0, BUF1(R1)              COPY 1 BYTE TO BUF1
            BXLE   R3, AGAIN                   IF (REG 3) = (REG 5), DONE
            •
            •
            •
BUF0        DS     10
BUF1        DS     10
```

Result of BXLE Instruction

Condition Code = Unchanged by BXLE Instruction
(REG3) = 0000000A
(REG4) = 00000001
(REG5) = 00000009

INSTRUCTION

Branch on Index High (BXH)

Assembler Notation		Op-Code	Format
BXH	R1,D2 (X2)	C0	RX1,RX2
BXH	R1,A2 (FX2,SX2)	C0	RX3

Set Up

R1	Starting index value
R1+1	Increment value
R1+2	Limit or final value

Prior to execution of this instruction, the register specified by R1 must contain a starting index value. The register specified by R1+1 must contain an increment value. The register specified by R1+2 must contain a comparand (limit or final value). All values may be signed.

Operation

Execution of this instruction causes the increment value to be added to the index value. The result is logically compared to the limit or final value. If the index value is greater than the limit value, a branch is executed to the second operand location. If the index value is equal to or less than the limit value, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The incremented index value replaces the contents of the register specified by R1.

The register numbers wrap around, i.e., three consecutive registers and by this instruction may be 6, 7, 8 or 14, 15, 0 or 15, 0, 1 etc.

The second operand location must be on a halfword boundary.

The branch address is calculated before incrementing the starting index value contained in the register specified by R1.

The register specified by R1 may be the same as X2, FX2 or SX2.

Example: BXH

The following example shows how to set up a counter (1 - 9) using the BXH instruction.

Label	Assembler Notation		Comment
	LIS	REG1,1	(REG 1) = 0000 0001 (INDEX)
	LIS	REG2,1	(REG 2) = 0000 0001 (INCREMENT)
	LIS	REG3,9	(REG 3) = 0000 0009 (COMPARAND)
BEGIN	BXH	REG1,LABEL	COMPARE INDEX WITH COMPARAND
	LH	R6,COUNT	
	.		
	.		
	.		
	B	BEGIN	BRANCH TO BXH INSTRUCTION
LABEL	LA	R8,RTN	EXIT FROM BXH
	ST	R8,MEM	

Result of BXH Instruction

Code between the instructions labelled BEGIN and LABEL will be executed 9 times.

Condition Code = Unchanged by BXH instruction
(REG1) = 0000 000A
(REG2) = 0000 0001
(REG3) = 0000 0009

EXTENDED BRANCH MNEMONICS

The CAL Assembler supports 14 extended branch mnemonics that generate the branch op-code (true or false conditional) and the condition code mask required. The programmer must supply the second operand address (symbolic or absolute). In the case of short format (SF) branch instructions, the second operand branch address must be within ± 15 halfwords of the current location counter. The CAL Assembler determines the backward or forward relationship of the second operand address and generates the appropriate operation code.

Examples of extended branch mnemonic:

```
                 .
                 .
                 .
                 LH          R5,L00P1
                 BNZ         LOERR
       LAP       SRLS        R6,1
                 BNCS        LAP
                 BS          CONTIN
       LOERR     LIS         R6,0
       ERROR1    AIS         R6,1
                 SIS         R5,4
                 BPS         ERROR1
                 SIS         R8,1
                 BO          ERROR2
       CONTIN    LH          R1,TIME
                 .
                 .
                 .
```

Appendix 4 lists the extended branch mnemonics and the proper operand form to be used with each mnemonic. The actual machine code generated is also listed.

The instructions described in this section are:

BC	Branch on Carry	BP	Branch on Plus
BCR	Branch on Carry Register	BPR	Branch on Plus Register
BCS	Branch on Carry Short	BPS	Branch on Plus Short
BNC	Branch on No Carry	BNP	Branch on Not Plus
BNCR	Branch on No Carry Register	BNPR	Branch on Not Plus Register
BNCS	Branch on No Carry Short	BNPS	Branch on Not Plus Short
BE	Branch on Equal	BO	Branch on Overflow
BER	Branch on Equal Register	BOR	Branch on Overflow Register
BES	Branch on Equal Short	BOS	Branch on Overflow Short
BNE	Branch on Not Equal	BNO	Branch on No Overflow
BNER	Branch on Not Equal Register	BNOR	Branch on No Overflow Register
BNES	Branch on Not Equal Short	BNOS	Branch on No Overflow Short
BL	Branch on Low	BZ	Branch on Zero
BLR	Branch on Low Register	BZR	Branch on Zero Register
BLS	Branch on Low Short	BZS	Branch on Zero Short
BNL	Branch on Not Low	BNZ	Branch on Not Zero
BNLR	Branch on Not Low Register	BNZER	Branch on Not Zero Register
BNLS	Branch on Not Low Short	BNZS	Branch on Not Zero Short
BM	Branch on Minus		
BMR	Branch on Minus Register	B	Branch (Unconditional)
BMS	Branch on Minus Short	BR	Branch Register (Unconditional)
		BS	Branch Short (Unconditional)
BNM	Branch on Not Minus		
BNMR	Branch on Not Minus Register		
BNMS	Branch on Not Minus Short	NOP	No Operation
		NOPR	No Operation Register

INSTRUCTION

Branch on Carry (BC)
Branch on Carry Register (BCR)
Branch on Carry Short (BCS)

Assembler Notation		Op-Code + M1	Format
BC	D2(X2)	428	RX1,RX2
BC	A2(FX2,SX2)	428	RX3
BCR	R2	028	RR
BCS	A	208 (Backward) 218 (Forward)	SF

Operation

If the Carry (C) flag in the Condition Code is set, a branch is taken to the second operand location. If the Carry flag is not set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

Example: BCS

Assembler Notation			Machine Code	Comments
SHIFT	SLLS	R9,1	1191	Register 9 is shifted left until the first zero bit is shifted out (left).
	BCS	SHIFT	2081	

INSTRUCTION

Branch on No Carry (BNC)
Branch on No Carry Register (BNCR)
Branch on No Carry Short (BNCS)

Assembler Notation		Op-Code + M1	Format
BNC	D2(X2)	438	RX1,RX2
BNC	A2(FX2,SX2)	438	RX3
BNCR	R2	038	RR
BNCS	A	228 (Backward)	SF
		238 (Forward)	

Operation

If the Carry (C) flag in the Condition Code is not set, a branch is taken to the second operand location. If the Carry flag is set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Equal (BE)
Branch on Equal Register (BER)
Branch on Equal Short (BES)

Assembler Notation		Op-Code + M1	Format
BE	D2(X2)	433	RX1,RX2
BE	A2(FX2,SX2)	433	RX3
BER	R2	033	RR
BES	A	223 (Backward) 233 (Forward)	SF

Operation

If the G flag and the L flag are both reset in the Condition Code, a branch is taken to the second operand location. If either flag is set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

Example: BE

Assembler Notation		Machine Code	Comments
CLHI	R4,X'23'	C540 0023	If R4 contains X'23', a branch is taken to location X'A00'. Otherwise the next sequential instruction is executed.
BE	OPTIN	4330 0A00	

INSTRUCTION

Branch on Not Equal (BNE)
Branch on Not Equal Register ((BNER)
Branch on Not Equal Short (BNES)

Assembler Notation		Op-Code + M1	Format
BNE	D2(X2)	423	RX1,RX2
BNE	A2(FX2,SX2)	423	RX3
BNER	R2	023	RR
BNES	A	203 (Backward) 213 (Forward)	SF

Operation

If the G flag or the L flag is set in the Condition Code, a branch is taken to the second operand location. If neither flag is set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Low (BL)
Branch on Low Register (BLR)
Branch on Low Short (BLS)

Assembler Notation		Op-Code + M1	Format
BL	D2(X2)	428	RX1,RX2
BL	A2(FX2,SX2)	428	RX3
BLR	R2	028	RR
BLS	A	208 (Backward) 218 (Forward)	SF

Operation

If the Carry (C) flag in the Condition Code is set, a Branch is taken to the second operand address. If the Carry flag is not set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

Example: BL

Assembler Notation		Machine Code	Comments
CLHI	R1,X'FF'	C510 00FF	R1 is compared to X'00FF'.
BL	RESTART	4280 0A00	If R1 is less than X'FF', a branch is taken to memory location X'0A00'.

INSTRUCTION

Branch on Not Low (BNL)
Branch on Not Low Register (BNLR)
Branch on Not Low Short (BNLS)

Assembler Notation		Op-Code + M1	Format
BNL	D2(X2)	438	RX1, RX2
BNL	A2(FX2,SX2)	438	RX3
BNLR	R2	038	RR
BNLS	A	228 (Backward) 238 (Forward)	SF

Operation

If the Carry (C) flag in the Condition Code is reset, a branch is taken to the second operand address. If the Carry flag is set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Minus (BM)
Branch on Minus Register (BMR)
Branch on Minus Short (BMS)

Assembler Notation		Op-Code + M1	Format
BM	D2(X2)	421	RX1,RX2
BM	A2(FX2,SX2)	421	RX3
BMR	R2	021	RR
BMS	A	201 (Backward) 211 (Forward)	SF

Operation

If the Less Than (L) flag in the Condition Code is set, a branch is taken to the second operand location. If the L flag is not set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

Example: BM

Assembler Notation		Machine Code	Comments
SIS	R3,1	2631	If R3 is less than 0 after the subtraction, a branch is taken to X'10A0'.
BM	CONTINUE	4210 10A0	

INSTRUCTION

Branch on Not Minus (BNM)
Branch on Not Minus Register (BNMR)
Branch on Not Minus Short (BNMS)

Assembler Notation		Op-Code + M1	Format
BNM	D2(X2)	431	RX1,RX2
BNM	A2(FX2,SX2)	431	RX3
BNMR	R2	031	RR
BNMS	A	221 (Backward) 231 (Forward)	SF

Operation

If the Less Than (L) flag in the Condition Code is reset, a branch is taken to the second operand location. If the L flag is set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Plus (BP)
Branch on Plus Register (BPR)
Branch on Plus Short (BPS)

Assembler Notation		Op-Code + M1	Format
BP	D2(X2)	422	RX1,RX2
BP	A2(FX2,SX2)	422	RX3
BPR	R2	022	RR
BPS	A	202 (Backward) 212 (Forward)	SF

Operation

If the Greater Than (G) flag in the Condition Code is set, a branch is taken to the second operand location. If the G flag is not set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Not Plus (BNP)
Branch on Not Plus Register (BNPR)
Branch on Not Plus Short (BNPS)

Assembler Notation		Op-Code + M1	Format
BNP	D2(X2)	432	RX1,RX2
BNP	A2(FX2,SX2)	432	RX3
BNPR	R2	032	RR
BNPS	A	222 (Backward) 232 (Forward)	SF

Operation

If the Greater Than (G) flag in the Condition Code is reset, a branch is taken to the second operand location. If the G flag is set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Overflow (BO)
Branch on Overflow Register (BOR)
Branch on Overflow Short (BOS)

Assembler Notation		Op-Code + M1	Format
BO	D2(X2)	424	RX1,RX2
BO	A2(FX2,SX2)	424	RX3
BOR	R2	024	RR
BOS	A	204 (Backward) 214 (Forward)	SF

Operation

If the Overflow (V) flag in the Condition Code is set, a branch is taken to the second operand location. If the V flag is reset, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on No Overflow (BNO)
Branch on No Overflow Register (BNOR)
Branch on No Overflow Short (BNOS)

Assembler Notation		Op-Code + M1	Format
BNO	D2(X2)	434	RX1,RX2
BNO	A2(FX2,SX2)	434	RX3
BNOR	R2	034	RR
BNOS	A	224 (Backward) 234 (Forward)	SF

Operation

If the Overflow (V) flag in the Condition Code is reset, a branch is taken to the second operand location. If the V flag is set, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Zero (BZ)
Branch on Zero Register (BZR)
Branch on Zero Short (BZS)

Assembler Notation		Op-Code + M1	Format
BZ	D2(X2)	433	RX1,RX2
BZ	A2(FX2,SX2)	433	RX3
BZR	R2	033	RR
BZS	A	223 (Backward) 233 (Forward)	SF

Operation

If the G and L flags are both reset in the Condition Code, a branch is taken to the second operand location. If the G or L flag is set, the next sequential instruction is executed.

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch on Not Zero (BNZ)
Branch on Not Zero Register (BNZR)
Branch on Not Zero Short (BNZS)

Assembler Notation		Op-Code + M1	Format
BNZ	D2(X2)	423	RX1,RX2
BNZ	A2(FX2,SX2)	423	RX3
BNZR	R2	023	RR
BNZS	A	203 (Backward) 213 (Forward)	SF

Operation

If the G or L flag in the Condition Code is set, a branch is taken to the second operand address. If the G and L flags are both reset, the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

INSTRUCTION

Branch (Unconditional) (B)
Branch Register (Unconditional) (BR)
Branch Short (Unconditional) (BS)

Assembler Notation		Op-Code + M1	Format
B	D2(X2)	430	RX1,RX2
B	A2(FX2,SX2)	430	RX3
BR	R2	030	RR
BS	A	220 (Backward) 230 (Forward)	SF

Operation

A branch is unconditionally taken to the second operand address.

Condition Code

Unchanged

Programming Note

The branch address must be located on a halfword boundary.

In the RR format, the branch address is contained in the register specified by R2.

Example: B

Assembler Notation		Machine Code	Comments
B	OPTIN	4300 0A00	An unconditional branch is taken to location X'0A00'.

INSTRUCTION

No Operation (NOP)
No Operation Register (NOPR)

Assembler Notation		Op-Code + M1	Format
NOP	D2(X2)	420	RX
NOPR	R2	020	RR

Operation

After a short delay (instruction decode time), the next sequential instruction is executed.

Condition Code

Unchanged

Programming Note

D2(X2) and R2 are ignored and usually equal zero (0).

Example: NOP, NOPR

Assembler Notation		Machine Code	Comments
NOP	0	4200 0000	No Operation
NOPR	0	0200	No Operation

CHAPTER 4
FIXED POINT ARITHMETIC

Fixed Point Arithmetic instructions provide a complete set of operations for calculating addresses and indexes, for counting, and for general purpose fixed point arithmetic.

DATA FORMATS

There are three formats for fixed point data: the halfword, the fullword, and the double word. In each of these formats, the most significant bit (Bit 0) is the Sign bit. The remaining bits, either 15, 31 or 63, represent the magnitude.

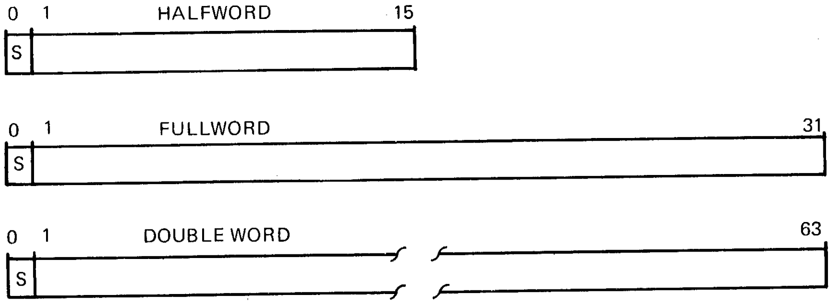


Figure 4-1. Fixed Point Data Words Formats

Positive values are represented in true binary form with a Sign bit of ZERO. Negative values are represented in two's complement form with a Sign bit of ONE. To change the sign of a number, the two's complement of the number is produced as follows:

1. Change all zeros to ones, and all ones to zeros.
2. Add one.

FIXED POINT NUMBER RANGE

Fixed point numbers represent integers. Table 4-1 shows relation between different formats along with decimal values.

TABLE 4-1. FIXED POINT FORMAT RELATIONS

DOUBLE WORD	FULLWORD	HALFWORD	DECIMAL
8000000000000000 (MOST NEGATIVE)			- 92233 72036 85477 5808
	80000000 (MOST NEGATIVE)		- 21474 83648
		8000 (MOST NEGATIVE)	- 32768
FFFFFFFFFFFFFFFF	FFFFFFFF	FFFF (LEAST NEGATIVE)	- 1
0000000000000000	00000000	0000	0
0000000000000001	00000001	0001	1
		7FFF (MOST POSITIVE)	32767
	7FFFFFFF (MOST POSITIVE)		21474 83647
7FFFFFFFFFFFFFFF (MOST POSITIVE)			92233 72036 85477 5807

OPERATIONS

The Fixed Point instructions include both fullword and halfword operations. Fullword operations take place between (a) the contents of two general registers, or (b) between the contents of a general register and a fullword stored in memory, or (c) between the contents of a general register and a fullword obtained from the instruction stream. Fullword multiply produces a double word result which is contained in two adjacent registers. Fullword divide operates on a double word contained in two adjacent registers.

Halfword operations take place between a fullword contained in one of the general registers and a halfword contained in memory. Before the operation is started, the halfword in memory is expanded to a fullword by propagating the most significant bit (Sign bit) into the high order bits of the fullword. (The Halfword Multiply and Divide instructions are exceptions to this rule.)

CONDITION CODE

All Fixed Point Arithmetic instructions except Multiply and Divide affect the Condition Code. The Condition Code indicates the effect of the operation on the 32 bit result.

In fixed point Add and Subtract operations, because the arguments are represented in two's complement form, all bits, sign included, participate in forming the result. Consequently, the occurrence of a carry or borrow has no real arithmetic significance.

For example, an Add operation between a minus one (FFFF FFFF) and a plus two (0000 0002) produces the correct result of plus one (0000 0001) and a carry. The Condition Code is set to 1010 (C = 1 and G = 1). "Carry only" means that the complete result, which in this case would have been 1 0000 0001, would not fit in 32 bits.

An overflow occurs when the result does not fit in 31 bits. Note that bit "zero" must be reserved for the sign of the result. For example, adding one to the largest positive fixed point value and will produce an overflow:

```
  7FFF FFFF
+ 0000  0001
= 8000   0000
```

the condition code is 0101 (V = 1 and L = 1)

The result, 8000 0000, is logically correct, but because the sign bit is negative when the result should be positive, the overflow condition exists.

The columns of the Condition Code table show the state of the C, V, G and L glags for the specific result.

The 'X' in the Condition Code column means that particular flag is not defined, i.e., the flag can be 0 or 1. Hence, no inference should be drawn by testing that particular flag.

FIXED POINT INSTRUCTION FORMATS

The fixed point instructions use the Register to Register (RR), the Short Form (SF), the Register and Indexed Storage (RX), and the Register and Immediate (RI) instruction formats.

FIXED POINT INSTRUCTIONS

The fixed point instructions described in this section are:

A	Add	CI	Compare Immediate
AR	Add Register	CH	Compare Halfword
AI	Add Immediate	CHI	Compare Halfword Immediate
AIS	Add Immediate Short	M	Multiply
AH	Add Halfword	MR	Multiply Register
AHI	Add Halfword Immediate	MH	Multiply Halfword
AM	Add to Memory	MHR	Multiply Halfword Register
AHM	Add Halfword to Memory	D	Divide
S	Subtract	DR	Divide Register
SR	Subtract Register	DH	Divide Halfword
SI	Subtract Immediate	DHR	Divide Halfword Register
SIS	Subtract Immediate Short	SLA	Shift Left Arithmetic
SH	Subtract Halfword	SLHA	Shift Left Halfword Arithmetic
SHI	Subtract Halfword Immediate	SRA	Shift Right Arithmetic
C	Compare	SRHA	Shift Right Halfword Arithmetic
CR	Compare Register	CHVR	Convert to Halfword Value Register

INSTRUCTIONS

Add (A)
Add Register (AR)
Add Immediate (AI)
Add Immediate Short (AIS)

Assembler Notation		Op-Code	Format
A	R1, D2 (X2)	5A	RX1, RX2
A	R1, A2 (FX2, SX2)	5A	RX3
AR	R1, R2	0A	RR
AI	R1, I2 (X2)	FA	RI2
AIS	R1, N	26	SF

Operation

The second operand is added algebraically to the contents of the register specified by R1. The result of this 32 bit addition replaces the contents of the register specified by R1.

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less then ZERO
X	0	1	0	Result is greater than ZERO
X	1	X	X	Arithmetic overflow
1	X	X	X	Carry

Programming Note

The second operand for the Add Immediate Short instruction is obtained by expanding the four bit data field, N, to a 32 bit fullword by forcing the high order bits to zero.

In the RX formats, the second operand must be located on a fullword boundary.

Example: A

Add contents of memory location labelled LAB to the contents of (REG) 4.

1. Register 4 Contains X'7F341234'
   Fullword in Memory at LAB contains X'7F124321'

Assembler Notation	Comments
A REG4, LAB	ADD (LAB) TO (REG 4)

Result of A Instruction

(REG4) = X'FE465555'
(LAB) = unchanged by this instruction
Condition Code = 1010 (C=1, G=1)

2. Register 5 Contains X'8000 0001'
   Fullword in memory at LAB contains X'80000002'

Assembler Notation	Comments
A REG5, LAB	ADD (LAB) TO (REG 5)

Result of A Instruction

(REG5) = X'00000003'
(LAB) = unchanged by this instruction
Condition Code = 1110 (C=1, V=1, G=1)

INSTRUCTIONS

Add Halfword (AH)
Add Halfword Immediate (AHI)

Assembler Notation		Op-Code	Format
AH	R1, D2 (X2)	4A	RX1, RX2
AH	R1, A2 (FX2, SX2)	4A	RX3
AHI	R1, I2 (X2)	CA	RI1

Operation

The 16 bit second operand is expanded to a 32 bit fullword by propagating the most significant bit through Bits 15:0 of the fullword. The fullword operand is added to the fullword contents of the register specified by R1. The result replaces the contents of the register specified by R1.

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO
X	1	X	X	Arithmetic overflow
1	X	X	X	Carry

Programming Note

In the RX formats, the second operand must be located on a halfword boundary.

Example: AH

This example adds the halfword at memory location labelled LAB to the contents of Register 4.

1. Register 4 contains X'00230002'
   Halfword at memory location LAB contains X'FFFF'

Assembler Notation		Comments
AH	REG4, LAB	ADD (LAB) TO (REG 4)

Result of AH Instruction

(REG4) = 00230001'
(LAB) = unchanged by this instruction
Condition Code = 1010 (C=1, G=1)

2. Register 5 contains X'FFFF FFF5'
   LAB contains X'FFF2'

Assembler Notation		Comments
AH	REG5, LAB	ADD LAB TO REG5

Result of AH Instruction

(REG5) = 'FFFF FFE7'
(LAB) = unchanged by this instruction
Condition Code = 1001 (C=1, L=1)

INSTRUCTION

Add to Memory (AM)

Assembler Notation		Op-Code	Format
AM	R1, D2 (X2)	51	RX1, RX2
AM	R1, A2 (FX2, SX2)	51	RX3

Operation

The fullword second operand is added algebraically to the contents of the register specified by R1. The result replaces the fullword second operand in memory. The contents of the register specified by R1 are not changed.

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO
X	1	X	X	Arithmetic overflow
1	X	X	X	Carry

Programming Note

The second operand must be located on a fullword boundary.

Example: AM

1. Add contents of register 8 to memory location labelled LOC:

Register 8 contains X'00000008'
Fullword in memory at LOC contains X'034289AB'

Assembler Notation	Comments
AM REG8, LOC	ADD (REG 8) TO (LOC)

Result of AM Instruction

(REG8) = X'00000008'
(LOC) = X'034289B3'
Condition Code = 0010 (G=1)

2. Add contents of register 7 to memory location labelled LOC:

Register 7 contains X'7F341234'
Fullword in memory at LOC contains X'7F124321'

Assembler Notation	Comments
AM REG7, LOC	ADD (REG 7) TO (LOC)

Result of AM Instruction

(REG7) = unchanged by this instruction
(LOC) = X'FE465555'
Condition Code = 0101 (V=1, L=1)

INSTRUCTION

Add Halfword to Memory (AHM)

Assembler Notation		Op-Code	Format
AHM	R1,D2 (X2)	61	RX1,RX2
AHM	R1,A2 (FX2,SX2)	61	RX3

Operation

The second operand is expanded to a fullword by propagating the most significant bit through Bits 15:0. This fullword is added algebraically to the contents of the register specified by R1. The 32 bit result is truncated to 16 bits by removing the most significant bits (Bits 0:15). The 16 bit result replaces the contents of the memory location specified by the effective address of the second operand. The contents of the register specified by R1 are not changed.

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO
X	1	X	X	Arithmetic overflow
1	X	X	X	Carry

Programming Note

The second operand must be located on a halfword boundary.

The Condition Code settings are based on the halfword result.

Example: AHM

This example adds the contents of Register 5 to the contents of memory location LAB.

1. Register 5 contains X'00230002'
Halfword in memory at LAB contains X'FFFF'

Assembler Notation	Comments
AHM REG5,LAB	ADD (REG 5) TO (LAB)

Result of AHM Instruction

(REG5) = unchanged by this instruction
(LAB) = 0001
Condition Code = 1010 (C=1,G=1)

2. Register 6 contains X'FFFF FFF5'
LAB contains X'FFF2'

Assembler Notation	Comments
AHM REG6,LAB	ADD (REG 6) TO (LAB)

Result of AHM Instruction

(REG6) = unchanged by this instruction
(LAB) = FFE7
Condition Code = 1001 (C=1, L=1)

INSTRUCTIONS

Subtract (S)
Subtract Register (SR)
Subtract Immediate (SI)
Subtract Immediate Short (SIS)

Assembler Notation		Op-Code	Format
S	R1, D2 (X2)	5B	RX1, RX2
S	R1, A2 (FX2, SX2)	5B	RX3
SR	R1, R2	0B	RR
SI	R1, I2 (X2)	FB	RI2
SIS	R1, N	27	SF

Operation

The fullword second operand is subtracted algebraically from the contents of the register specified by R1. The result replaces the contents of the register specified by R1.

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO
X	1	X	X	Arithmetic overflow
1	X	X	X	Borrow

Programming Note

The second operand for the Subtract Immediate Short instruction is obtained by expanding the four bit data field, N, to a 32 bit fullword by forcing the high order bits to zero.

In the RX formats, the second operand must be located on a fullword boundary.

Examples:

This example subtracts the fullword at memory location LOC from the contents of Register 9.

1. REG9 contains X'44444444'
   LOC contains X'44444444'

Assembler Notation	Comments
S REG9, LOC	Subtract contents of (LOC) from (REG 9)

Result of S Instruction

(REG9) = 0

LOC = X'44444444'

Condition Code = 0000

2. REG9 contains X'23456789'
   LOC contains X'FFFF4321'

Assembler Notation	Comments
S REG9, LOC	Subtract contents of (LOC) from (REG 9)

Result of S Instruction

(REG9) = 23462368

(LOC) = FFFF4321

Condition Code = 1010 (C=1, G=1)

INSTRUCTIONS

Subtract Halfword (SH)
Subtract Halfword Immediate (SHI)

Assembler Notation		Op-Code	Format
SH	R1, D2 (X2)	4B	RX1, RX2
SH	R1, A2 (FX2, SX2)	4B	RX3
SHI	R1, I2 (X2)	CB	RI1

Operation

The 16 bit second operand is expanded to a 32 bit fullword by propagating the most significant bit through Bits 15:0. This fullword is subtracted from the contents of the register specified by R1. The result replaces the contents of the register specified by R1.

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO
X	1	X	X	Arithmetic overflow
1	X	X	X	Borrow

Programming Note

In the RX formats, the second operand must be located on a halfword boundary.

Example: SH

This example subtracts the halfword at memory location LOC from the contents of register 9.

1. REG9 contains X'00123456'

   LOC contains X'FFF4'

Assembler Notation		Comments
SH	REG9, LOC	Subtract contents of (LOC) from (REG 9)

Result of SH Instruction

(REG9) = 00123462
(LOC) = FFF4
Condition Code = 1010

2. REG9 contains X'FFFF4567'
   LOC contains X'2345'

Assembler Notation		Comments
SH	REG9, LOC	Subtract contents of (LOC) from (REG 9)

Result of SH Instruction

(REG9) = FFFF2222
(LOC) = 2345
Condition Code = 0001

INSTRUCTIONS

Compare (C)
Compare Register (CR)
Compare Immediate (CI)

Assembler Notation		Op-Code	Format
C	R1, D2 (X2)	59	RX1, RX2
C	R1, A2 (FX2, SX2)	59	RX3
CR	R1, R2	09	RR
CI	R1, I2 (X2)	F9	RI2

Operation

The first operand, contained in the register specified by R1, is compared algebraically to the 32 bit second operand. The result is indicated by the Condition Code setting. Neither operand is changed.

Condition Code

C	V	G	L	
0	X	0	0	First operand is equal to second operand
1	X	0	1	First operand is less than second operand
0	X	1	0	First operand is greater than second operand

Programming Note

In the RX formats, the second operand must be located on a fullword boundary.

The state of the V flag is undefined.

Example: C

This example compares the contents of Register 3 to the contents of the fullword in memory location LAB.

Register 3 contains X'44567894'
Fullword at LAB contains X'04321243'

Assembler Notation	Comments
C REG3, LAB	Compare (REG 3) to (LAB)

Result of C Instruction

(REG3) = unchanged by this instruction
(LAB) = unchanged by this instruction
Condition Code = 0010 (G=1)

INSTRUCTIONS

Compare Halfword (CH)
Compare Halfword Immediate (CHI)

Assembler Notation		Op-Code	Format
CH	R1,D2 (X2)	49	RX1,RX2
CH	R1,A2 (FX2,SX2)	49	RX3
CHI	R1,I2 (X2)	C9	RI1

Operation

The halfword second operand is expanded to a fullword by propagating the most significant bit through Bits 15:0. This fullword is compared algebraically with the first operand, the contents of the register specified by R1. The result is indicated by the Condition Code setting. Neither operand is changed.

Condition Code

C	V	G	L	
0	X	0	0	First operand is equal to second operand
1	X	0	1	First operand is less than second operand
0	X	1	0	First operand is greater than second operand

Programming Note

In the RX formats, the second operand must be located on a halfword boundary.

Condition code settings are based on the fullword comparison. The state of the V flag is undefined.

Example: CH

This example compares the contents of REG8 to the halfword at LAB.

REG8 contains X'F4567891'
Halfword at LAB contains X'3123'

Assembler Notation		Comments
CH	REG8, LAB	Compare (REG 8) to (LAB)

Result of CH Instruction

(REG8) = unchanged by this instruction
(LAB) = unchanged by this instruction
Condition Code = 1001 (C=1, V=1)

INSTRUCTIONS

Multiply (M)
Multiply Register (MR)

Assembler Notation		Op-Code	Format
M	R1, D2 (X2)	5C	RX1, RX2
M	R1, A2 (FX2, SX2)	5C	RX3
MR	R1, R2	1C	RR

Operation

The fullword first operand, contained in the register specified by R1 ~~+~~ 1, is multiplied by the fullword second operand. The 64 bit result is stored in the registers specified by R1 and R1 + 1.

Condition Code

Unchanged

Programming Note

The R1 field of these instructions must specify an even numbered register.

If R1 field of these instructions is odd, the result is undefined.

In the RX formats, the second operand must be located on a fullword boundary.

The most significant bits of the result are placed in the register specified by R1, the least significant bits of the result are placed in the register specified by R1 + 1.

The sign of the result is determined by the rules of algebra.

Example: M

This example multiplies the contents of Register 9 by the contents of memory location LOC and places the answer in Registers 8 and 9 (64 bits).

REG9 contains X'00002431'
Fullword at location LOC contains X'43120000'

Assembler Notation	Comments
M REG8, LOC	Multiply (REG 9) by (LOC)

Result of M Instruction

REG8 and REG9 together contain the answer
(REG8, REG9) = 0000 097B, 5E72 0000
(LOC) = unchanged by this instruction
Condition Code = unchanged by this instruction

Multiply Halfword (MH)
Multiply Halfword Register (MHR)

Assembler Notation		Op-Code	Format
MH	R1,D2 (X2)	4C	RX1,RX2
MH	R1,A2 (FX2,SX2)	4C	RX3
MHR	R1,R2	0C	RR

Operation

The first operand, contained in Bits 16:31 of the register specified by R1, is multiplied by the 16 bit second operand, taken from memory or from Bits 16:31 of the register specified by R2. The 32 bit result replaces the contents of the register specified by R1.

Condition Code

Unchanged

Programming Note

In the RX formats, the second operand must be located on a halfword boundary.

The sign of the result is determined by the rules of algebra.

Example: MH

This example multiplies the halfword contents of Register 8 by the halfword in memory location LAB.

REG8 contains X'ABCD 0045'
Halfword at memory location LAB contains X'8674'

Assembler Notation	Comments
MH REG8, LAB	Multiply least significant half of (REG 8) by (LAB)

Result of MH Instruction

(REG8) = FFDF3D44
(LAB) = unchanged by this instruction
Condition Code = unchanged by this instruction

Divide (D)
Divide Register (DR)

Assembler Notation		Op-Code	Format
D	R1, D2 (X2)	5D	RX1, RX2
D	R1, A2 (FX2, SX2)	5D	RX3
DR	R1, R2	1D	RR

Operation

The 64 bit dividend contained in the register specified by R1 and the register specified by R1+1 is divided by the fullword divisor. The 32 bit signed remainder replaces the contents of the register specified by R1. The 32 signed bit quotient replaces the contents of the register specified by R1+1.

Condition Code

Unchanged

Programming Note

The R1 field of these instructions must specify an even numbered register. If the R1 field of these instructions is odd, the result is undefined.

The most significant bits of the dividend must be contained in the register specified by R1. The least significant bits of the dividend must be contained in the register specified by R1 + 1.

In the RX formats, the second operand must be located on a fullword boundary.

If the divisor is equal to zero, the instruction is not executed, the operand registers are unchanged, and the arithmetic fault interrupt is taken, if enabled by Bit-19 of the current program status word. If the interrupt is not enabled, the next sequential instruction is executed.

If the value of the quotient is greater than X'7FFFFFFF' or less than (more negative than) X'80000000', quotient overflow is said to occur. If quotient overflow occurs, the operand registers are not changed, and the arithmetic fault interrupt is taken, if enabled by Bit-19 of the current program status word. If the interrupt is not enabled, the next sequential instruction is executed.

The sign of the quotient is determined by the rules of algebra. The sign of the remainder is same as the sign of the dividend.

Example: D

This example divides the contents of Registers 8 and 9 by the fullword contents of memory location LOC.

1. REG8 contains X'12345678' = First Half of Dividend
   REG9 contains X'98765432' = Second Half of Dividend
   LOC contains X'34343434' = Divisor

Assembler Notation		Comments
D	REG8, LOC	Divide (REG 8, 9) by (LOC)

Result of D Instruction

(REG8) 1E1E1E1E = Remainder
(REG9) = 59455459 = Quotient
(LOC) = 34343434
Condition Code = unchanged by this instruction

2. REG8 contains X'FFFF1234' = First Half of Dividend
REG9 contains X'00000000' = Second Half of Dividend
LOC contains X'12345678' = Divisor

Assembler Notation	Comments
D REG8, LOC	Divide (REG 8,9) by (LOC)

Result of D Instruction

(REG8) = F250D9E0 = Remainder
(REG9) = FFF2EFFC = Quotient
LOC = 12345678
Condition Code = unchanged by this instruction

3. REG8 contains X'43657898' = First Half of Dividend
REG9 contains X'12123456' = Second Half of Dividend
LOC contains X'00000000' = Divisor

Assembler Notation	Comments
D REG8, LOC	Divide (REG8, 9) by (LOC)

Result of D Instruction

Division by zero causes arithmetic fault to be taken if Bit 19 of PSW is enabled.

Operands and Condition Code remain unchanged by this instruction.

4. REG8 contains X'80000000' = First Half of Dividend
REG9 conatins X'00000001' = Second Half of Dividend
LOC contains X'00000001' = Divisor

Assembler Notation	Comments
D REG8, LOC	Divide (REG 8,9) by (LOC)

Result of D Instruction

Quotient overflow causes arithmetic fault to be taken if Bit-19 of PSW is enabled.

Operands and Condition Code remain unchanged by this instruction.

Divide Halfword (DH)
Divide Halfword Register (DHR)

Assembler Notation		Op-Code	Format
DH	R1, D2 (X2)	4D	RX1, RX2
DH	R1, A2 (FX2, SX2)	4D	RX3
DHR	R1, R2	0D	RR

Operation

The 32 bit dividend contained in the register specified by R1 is divided by a 16 bit divisor taken from memory or from Bits 16:31 of the register specified by R2. The 16 bit remainder is expanded to a fullword by propagating the Sign bit through Bits 15:0 and is stored in the register specified by R1. The 16 bit quotient is expanded to a fullword by propagating the Sign bit through Bits 15:0 and is stored in the register specified by R1+1.

SIGN OF REM. SAME AS SIGN OF DIVIDND.

Condition Code

Unchanged

Programming Note

In the RX formats, the second operand must be located on a halfword boundary.

If the divisor is equal to zero, the instruction is not executed, the operand registers are unchanged, and the arithmetic fault interrupt is taken, if enabled by Bit-19 of the current program status word. If the interrupt is not enabled, the next sequential instruction is executed.

If the value of the quotient is greater than X'7FFF' or less than (more negative than) X'8000', quotient overflow is said to occur.

If quotient overflow occurs, the operand registers are not changed, and the arithmetic fault interrupt is taken, if enabled by the Bit-19 of the current program status word. If the interrupt is not enabled, the next sequential instruction is executed.

The sign of the quotient is determined by the rules of algebra.

The sign of the remainder is same as the sign of the dividend.

Example: DH

This example divides the halfword contents of memory location LOC into the contents of Register 7.

1. REG7 contains X'0000 0054' = Dividend
   LOC contains X'0008' = Divisor

Assembler Notation	Comments
DH REG7, LOC	Divide (REG 7) by (LOC)

Result of DH Instruction

(REG7) = 0000 0004 = Remainder
(REG8) = 0000 000A = Quotient
(LOC) = 0008
Condition Code = unchanged by this instruction

2. REG7 contains X'12345678' = Dividend
LOC contains X'0000' = Divisor

Assembler Notation	Comments
DH REG7, LOC	Divide (REG 7) by (LOC)

Result of DH Instruction

Division by zero causes arithmetic fault to be taken if Bit-19 of PSW is enabled.

Operands and Condition Code remain unchanged by this instruction.

3. REG7 contains X'8000 0002' = Dividend
LOC contains X'0001'

Assembler Notation	Comments
DH REG7, LOC	Divide (REG 7) by (LOC)

Result of DH Instruction

Quotient overflow causes arithmetic fault to be taken if Bit-19 of PSW is enabled.

Operands and Condition Code remain unchanged by this instruction.

INSTRUCTION

Shift Left Arithmetic (SLA)

Assembler Notation		Op-Code	Format
SLA	R1, I2 (X2)	EF	RI1

Operation

Bits 1:31 of the first operand, contained in the register specified by R1, are shifted left the number of places specified by the second operand. The Sign bit (Bit 0), remains unchanged. Bits shifted out of Position 1 are shifted through the carry flag and then lost. The last bit shifted remains in the carry flag. Zeros are shifted into Position 31.

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO

Programming Note

The state of the C flag indicates the state of the last bit shifted.

The shift count is specified by the least significant five bits of the second operand.

A shift of zero places causes the Condition Code to be set in accordance with the value contained in the register specified by R1. The state of the C flag is undefined in this case.

Example: SLA

This example shifts the bits in Register 5 left by the number specified by the second operand.

REG5 contains X'80005647'

Assembler Notation	Comments
SLA REG5, 4	Shift Left 4 Places

Result of SLA Instruction

(REG5) = 80056470
Condition Code = 0001 (L=1)

INSTRUCTION

Shift Left Halfword Arithmetic (SLHA)

Assembler Notation		Op-Code	Format
SLHA	R1, I2 (X2)	CF	RI1

Operation

Bits 17:31 of the register specified by R1 are shifted left the number of places specified by the second operand. Bit 16 of the register, the halfword Sign bit, remains unchanged. Bits shifted out of Position 17 are shifted through the carry flag and then lost. The last bit shifted remains in the carry flag. Zeros are shifted into Position 31. Bits 0:15 of the first operand register remain unchanged.

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO

Programming Note

The Condition Code settings are based on the halfword, Bits 16:31, result.

The state of the C flag indicates the state of the last bit shifted.

The shift count is specified by the least significant four bits of the second operand.

A shift of zero places causes the Condition Code to be set in accordance with the halfword value contained in Bits 16:31 of the register. The state of the C flag is undefined in this case.

INSTRUCTION

Shift Right Arithmetic (SRA)

Assembler Notation		Op-Code	Format
SRA	R1, I2 (X2)	EE	RI1

Operation

Bits 1:31 of the first operand, contained in the register specified by R1, are shifted right the number of places specified by the second operand. The Sign bit (Bit 0), remains unchanged and is propagated right as many positions as specified by the second operand. Bits shifted out of Position 31 are shifted through the carry flag and lost. The last bit shifted remains in the carry flag.

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO

Programming Note

The state of the C flag indicates the state of the last bit shifted.

The shift count is specified by the least significant five bits of the second operand.

A shift of zero places causes the Condition Code to be set in accordance with the value contained in the register. The state of the C flag in undefined in this case.

Example: SRA

This example shifts the contents of Register 9 right the number of places specified by the second operand.

REG9 contains X'800004256'

Assembler Notation	Comments
SRA REG9, 8	Shift (REG 9) right 8 bits

Result of SRA Instruction

(REG9) = X'FF800042'
Condition Code = 0001 (L=1)

INSTRUCTION

Shift Right Halfword Arithmetic (SRHA)

Assembler Notation		Op-Code	Format
SRHA	R1, I2 (X2)	CE	RI1

Operation

Bits 17:31 of the register specified by R1 are shifted right the number of places specified by the second operand. Bit-16 of the register, the halfword Sign bit, remains unchanged and is propagated right the number of positions specified by the second operand. Bits shifted out of Position 31 are shifted through the carry flag and lost. The last bit shifted remains in the carry flag. Bits 0:15 of the first operand register remain unchanged.

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO

Programming Note

The condition code settings are based on the halfword, Bits 16:31, result.

The state of the C flag indicates the state of the last bit shifted.

If the second operand specifies a shift of zero places, the state of the C flag is undefined.

INSTRUCTION

Convert to Halfword Value Register (CHVR)

Assembler Notation	Op-Code	Format
CHVR R1,R2	12	RR

Operation

The halfword second operand, (Bits 16:31) of the register specified by R2, is expanded to a fullword by propagating the most significant bit (Bit 16) through Bits 15:0. This fullword replaces the contents of the register specified by R1.

Condition Code

C	V	G	L	
X	X	0	0	Result is ZERO
X	X	0	1	Result is less than ZERO
X	X	1	0	Result is greater than ZERO
X	1	X	X	Source operand cannot be represented by a 16 bit signed number
1	X	X	X	Carry flag was set in previous Condition Code
0	X	X	X	Carry flag was reset in previous Condition Code

Programming Note

The V flag is set when Bits 0:15 of the second operand are not the same as Bit-16 of the second operand. (In this case, the G and L flags reflect the algebraic value of Bits 16:31 of the second operand.)

Execution of this instruction following halfword operations guarantees results identical with those that would be obtained if the program were run on an INTERDATA 16 bit machine. For example, assume that location A in memory contains the halfword value of X'7FFF' (decimal 32767) then,

```
LH     R1,A         R1 contains X'00007FFF'
AIS    R1,1         R1 contains X'00008000'
```

Following the add operation, the Condition Code is:

C	V	G	L
0	0	1	0

indicating a result greater than zero, which is correct for fullword operations. If the same sequence were executed on a 16 bit Processor, as:

```
LH     R1,A         R1 contains X'7FFF'
AIS    R1,1         R1 contains X'8000'
```

Following this, the Condition Code in the halfword Processor is:

C	V	G	L
0	1	0	1

indicating overflow and a negative result. Going back to the original sequence and adding the Convert to Halfword Value instruction produces the following:

```
LH      R1,A         R1 contains X'00007FFF'
AIS     R1,1         R1 contains X'00008000'
CHVR    R1,R1        R1 contains X'FFFF8000'
```

Following this sequence, the Condition Code is:

C	V	G	L
0	1	0	1

which is identical to that of the 16 bit Processor, and can be tested in the same manner.

CHAPTER 5
FLOATING POINT ARITHMETIC

Floating Point Arithmetic instructions provide a means for rapid manipulation of scientific data expressed as floating point numbers. Single Precision as well as Double Precision Floating Point Instructions are described in this chapter. The comprehensive set of instructions includes load and store floating point numbers; add, subtract, multiply, divide and compare two floating point numbers; convert fixed point to floating point and vice versa.

INTRODUCTION

Floating point is a means of respresenting a quantity in any numbering system. Consider a decimal number (base = 10), 123 which can be represented in the following forms:

$$\begin{array}{rl} 123.0 & \times\ 10^{0} \\ 1.23 & \times\ 10^{2} \\ 0.123 & \times\ 10^{3} \\ 0.0123 & \times\ 10^{4} \end{array}$$

Note that in this example, the decimal point moved. Hence we have a floating point. In actual floating point representation the significant digits are always fractional and are collectively referred to as fraction. The power to which the base number is raised is called the exponent. For example, in the number "$.45678 \times 10^{2}$", 45678 is the fraction and 2 is the exponent. Both the fraction and the exponent may be signed. If we have a floating point representation as,

(sign of fraction) (exponent) (fraction)

then the following representation applies:

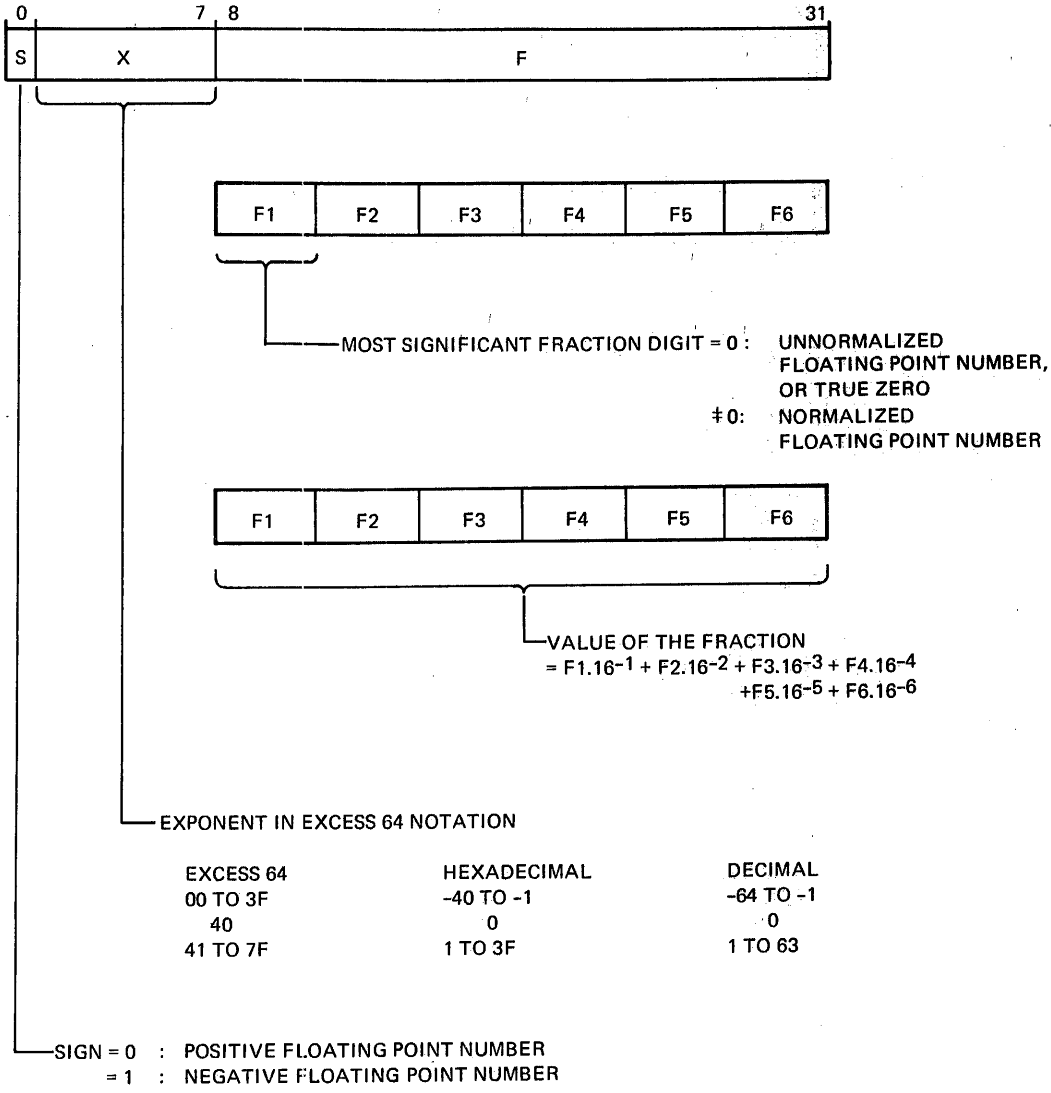
Number		Floating point Sign	Exponent	Fraction
+32.94	$= +.3294 \times 10^{2}$	+	+2	3294
-23760000.0	$= -.2376 \times 10^{8}$	-	+8	2376
+0.000059	$= +.59 \times 10^{-4}$	+	-4	59
-0.0000000092073	$= -.92073 \times 10^{-8}$	-	-8	92073

The convenience with which extremely large or small numbers can be expressed in floating point makes it ideally suitable for scientific computation. Note the compactness of floating point notation in the above examples.

The floating point representation in the Model 7/32 is similar to the above representation. The differences are as follows:

Hexadecimal, instead of decimal, numbering system is used.
Physical size of the number and hence the magnitude and the precision is limited.

The single precision floating point number fields are shown in Figure 5-1.


0
7
8
31
S
X
F
F1
F2
F3
F4
F5
F6
MOST SIGNIFICANT FRACTION DIGIT = 0 : UNNORMALIZED FLOATING POINT NUMBER, OR TRUE ZERO
‡0: NORMALIZED FLOATING POINT NUMBER
F1
F2
F3
F4
F5
F6
VALUE OF THE FRACTION
$= F1.16^{-1} + F2.16^{-2} + F3.16^{-3} + F4.16^{-4} + F5.16^{-5} + F6.16^{-6}$
EXPONENT IN EXCESS 64 NOTATION
SIGN = 0 : POSITIVE FLOATING POINT NUMBER
= 1 : NEGATIVE FLOATING POINT NUMBER

EXCESS 64	HEXADECIMAL	DECIMAL
00 TO 3F	-40 TO -1	-64 TO -1
40	0	0
41 TO 7F	1 TO 3F	1 TO 63

Figure 5-1. Single Precision Floating Point Number Fields

FLOATING-POINT NUMBER

In the Model 7/32 Processor a floating point number is represented in the following form:

Sign	Exponent	Fraction

Sign — The most significant bit of a floating point number is a sign bit. The sign bit is zero for positive numbers and one for negative numbers. The floating point value of zero always has a positive sign.

Exponent — The 7-Bit field, Bits 1:7, is designated as the exponent field. The exponent field contains the true value of the exponent plus X'40' (decimal 64). This helps to represent very small magnitudes between 0 and 1. The exponent is said to be expressed in excess 64 notation. Some of the exponent values are as follows:

Exponent in Excess 64 notation	True exponent in hexadecimal	True exponent in decimal	Multiply fraction by
00	-40	-64	16^{-64}
3F	-1	-1	16^{-1}
40	0	0	1
41	1	1	16
7F	3F	63	16^{+63}

The exponent field for true zero is always 00.

Fraction — The fraction field is 6-hexadecimal digits for single precision floating point numbers (thus limiting the precision) and 14-hexadecimal digits for double precision floating point numbers. As in any other fraction, the floating point fraction is expressed with most precision when the most significant digit (not necessarily the most significant bit) is non-zero. The floating point number with such a fraction is called a normalized floating point number. In the model 7/32 Processor, normalized numbers are always used to obtain maximum possible precision. For hexadecimal fraction conversion, refer to Appendix 6.

Examples: The following examples illustrate the sign, exponent and fraction concept of a floating point number.

Numbers in Hex integer-fraction notation	Sign-exponent-fraction shown for clarity			Single Precision Floating point numbers
	S	E	F	
+1.3A25678	0	41	13A25678	4 1 1 3 A 2 5 6
-6.89F2C	1	41	689F2C	C 1 6 8 9 F 2 C
+1A.C39D21	0	42	1AC39021	4 2 1 A C 3 9 D
-3C1DF.82A3	1	45	3C1DF82A3	C 5 3 C 1 D F 8
+ABCDEF12.9AC	0	48	ABCDEF129AC	4 8 A B C D E F
+0.0032A9CF2	0	3E	32A9CF2	3 E 3 2 A 9 C F
-0.000002C7B5	1	3B	2C7B5	B B 2 C 7 B 5 0

Refer to Appendix 6 for examples of similar conversion to double precision floating point numbers.

Floating Point Number Range

The range of magnitude (M) of a normalized floating point number is as follows.

Single precision:	$16^{-65} \leq M \leq (1 - 16^{-6}) * 16^{63}$
Double precision:	$16^{-65} \leq M \leq (1 - 16^{-14}) * 16^{63}$
Approximately for both:	$5.4 * 10^{-79} < M < 7.2 * 10^{75}$

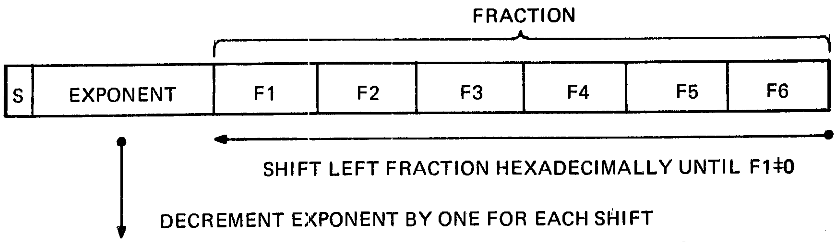
Table 5-1 shows the single precision point range in relocation to the fixed point range along with the decimal values.

TABLE 5-1 FLOATING/FIXED POINT RANGES

Floating Point numbers	Fixed Point integer	Decimal numbers
(most negative) FFFF FFFF		$-7.2 * 10^{75}$
C880 0000	8000 0000 (most negative)	-2 147 483 648
C111 0000	FFFF FFFF (least negative)	-1
(least negative) 8010 0000		$-5.4 * 10^{-79}$
(true zero) 0000 0000	0000 0000	0
(least positive) 0010 0000		$+5.4 * 10^{-79}$
4110 0000	0000 0001 (least positive)	+1
487F FFFF	7FFF FFFF (most positive)	+2 147 483 647
(most positive) 7FFF FFFF		$+7.2 * 10^{75}$

Normalization

Normalization is a process of making non-zero the most significant digit (F1) of the fraction of a floating point number. In the normalization process, the floating point fraction is shifted left hexadecimally (i.e., four bits at a time), and its exponent is decremented by one for each hexadecimal shift until the most significant digit (not necessarily the most significant bit) of the fraction is non-zero.


FRACTION
S
EXPONENT
F1
F2
F3
F4
F5
F6
SHIFT LEFT FRACTION HEXADECIMALLY UNTIL F1≠0
DECREMENT EXPONENT BY ONE FOR EACH SHIFT

Except LE, LER, LD, LDR instructions, all the floating point operations assume and require normalized operands for consistent results. The LE, LER, LD and LDR instructions normalize an unnormalized operand.

Example:

	Operands	After normalization
1.	4 2 0 1 2 3 4 5	4 1 1 2 3 4 5 0
2.	2 1 0 0 0 A B C	1 E A B C 0 0 0
3.	C 9 0 0 F E 1 2	C 7 F E 1 2 0 0
4.	6 C 0 0 0 0 0 0	0 0 0 0 0 0 0 0 (true zero)
5.	8 2 0 0 0 A 6 7	0 0 0 0 0 0 0 0 (exponent underflow)

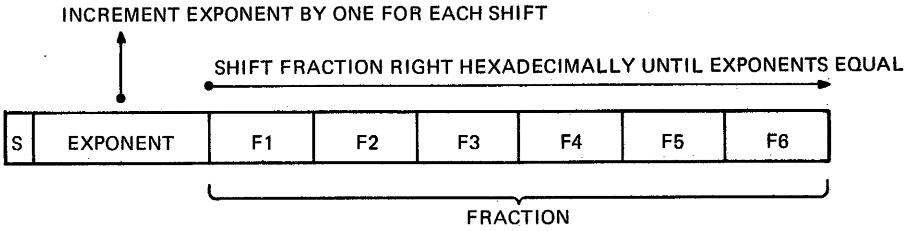
In example 4, the fraction of the operand is zero. During the normalization process, such a fraction is detected and the floating point number is set to true zero.

In example 5, the exponent of the operand is very small. During the normalization process, the exponent is decremented from 00 to 7F. Such a transition results in exponent underflow and the floating point number is set to true zero.

In floating point operations, assuming that the operands are normalized, normalized results are always produced. Results of operations between unnormalized numbers are undefined. Results of operation between unnormalized numbers are undefined.

Equalization

Equalization is a process of making equal the exponents of two floating point numbers. The fraction of the floating point number with the smaller exponent is shifted right hexadecimally, i.e., four bits at a time, and its exponent is incremented by one for each hexadecimal shift until the two exponents are equal.


INCREMENT EXPONENT BY ONE FOR EACH SHIFT
SHIFT FRACTION RIGHT HEXADECIMALLY UNTIL EXPONENTS EQUAL
S
EXPONENT
F1
F2
F3
F4
F5
F6
FRACTION

During the floating point addition and subtraction two floating point operands are equalized.

Example:

	Floating point operands	After equalization
1.	43123456 3F789ABC	43123456 43000078
2.	C7FE1234 4956789A	C900FE12 4956789A

In this example, normalized floating point numbers are shown because addition and subtraction require normalization. Note that if the exponents differ by 6 or more the significance of the lower exponent floating point number is lost in the process of equalization.

True Zero

A floating point number is said to be true zero when the exponent and the fraction fields are all zeroes. In other words, all data bits must be zero. A value of zero always has a positive sign. In general, zero values participate as normal operands in all floating point operations.

A true zero may be used as an operand or may result from an arithmetic operation that caused an exponent underflow, in which case the entire number is forced to true zero. Secondly, if an arithmetic operation produces a result whose fraction digits are all zeroes (sometimes referred to as loss of significance), the entire number is forced to true zero.

Examples:

Numbers	Operation	Result	Reason
030000AB	Normalize	0000 0000	exponent underflow
41ABCDEF 41ABCDEF	Subtract	0000 0000	loss of significance

Exponent Overflow

In floating point operations, exponent overflow may occur. Exponent overflow occurs when a resulting exponent is greater than +63. If overflow occurs, the exponent and fraction bits of the result are set to all 1s, the largest possible magnitude and therefore the closest possible answer. The sign of the result is not affected by the overflow. Figure 5-2 illustrates exponent overflow using a line representation of numbers.

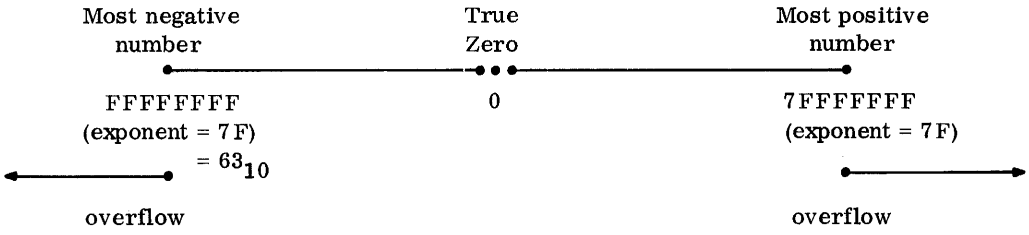


Figure 5-2. Exponent Overflow

If overflow occurs, the V flag in the Condition Code is set, and an arithmetic fault interrupt is taken, if enabled by the current PSW.

Exponent Underflow

The normalization process, during a floating point operation, may produce an exponent underflow. Exponent underflow occurs when a result exponent would be less than -64. If underflow occurs, the entire result is set to true zero, the closest possible answer. Figure 5-3 illustrates exponent underflow using a line representation of numbers.

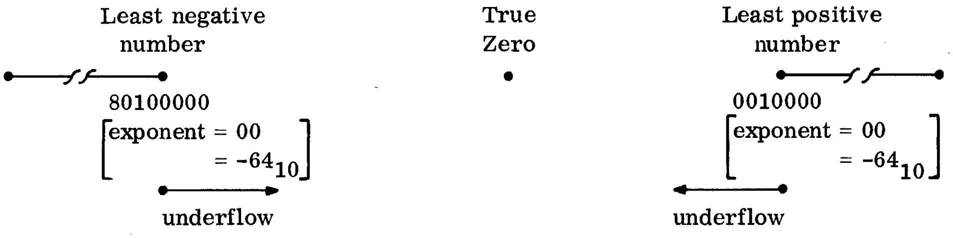


Figure 5-3. Exponent Underflow

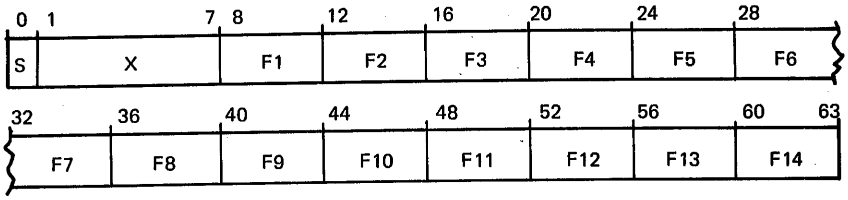
If underflow occurs, the V flag in the Condition Code is set, and an arithmetic fault interrupt is taken, if enabled by the current PSW.

Data Formats

In the Model 7/32 Processor, floating point numbers occur in one of two formats, single precision and double precision. The single precision format requires a fullword (32 bits) in one of the 8 single precision floating point registers or on a fullword address boundary, in memory. The sign (s), exponent (x) and fraction (consisting of digits F1, F2, F3, F4, F5 and F6) fields are designated as follows:

0	1 7	8	12	16	20	24	28 31
S	X	F1	F2	F3	F4	F5	F6

The double precision format requires a doubleword (64 bits) in one of the 8 double precision floating point registers or on a doubleword address boundary in memory. The sign (s), exponent (x) and fraction (consisting of digits F1 through F14) fields are designated as follows:



The value of a single (and similarly double) precision floating point number can be expressed as follows:

$$\text{sign}(F1.16^{-1} + F2.16^{-2} + F3.16^{-3} + F4.16^{-4} + F5.16^{-5} + F6.16^{-6})\ 16^{X-x'40'}$$

Guard Digit and Rounding

A guard digit is an extra hexadecimal digit provided to the right of the least significant fraction digit of a floating point number. In the Model 7/32, only single precision floating point numbers can have a guard digit. The guard digit is produced and used during the processing of intermediate results of a floating point operation. The guard digit does not appear in the final result. However, the guard digit helps rounding the final result, thus increasing the precision slightly. In the absence of a guard digit, as is the case in double precision floating point numbers, the final result is simply truncated.

NOTE

> Some of the earlier models of the 7/32 Processor, which do not have the double precision floating point option, do not have a guard digit for single precision floating point numbers. Hence the results are truncated, not rounded.

A guard digit is produced during the equalization phase of an Add and Subtract single precision floating point operation. Then the operation is performed to obtain an intermediate result. The guard digit participates in the operation. If the guard digit of the intermediate result is 0 through 7, no rounding is done. If it is 8 through F, one (1) is added to the fraction of the intermediate result to obtain the final result fraction, unless such an addition produces a carry into the exponent field. The following example illustrates the rounding procedure.

```
                      After          Guard
  operands         equalization      digit
                                       ↓
   42ABCD12       4 2 A B C D 1 2 [0]
 + 416789AB     + 4 2 0 6 7 8 9 A [B]
                ___________________
                  4 2 B 2 4 5 A C [B]    intermediate result
                +               1
                ________________
                  4 2 B 2 4 5 A D        final result
```

A guard digit is also produced during the Multiply and Divide single precision floating point operations. The intermediate product or the quotient is rounded as shown here to obtain the final result.

Conversion from Decimal

The process of converting a decimal number into the excess 64 notation used internally by the Processor involves the following steps:

1. Separate the decimal integer from the decimal fraction:

   $182.375_{10} = (182 + .375)_{10}$

2. Convert each part to hexadecimal by referring to the Integer conversion table and the Fraction conversion table in Appendix 5.

   $182_{10} = B6_{16}$ $\quad .375_{10} = .6_{16}$

3. Combine the hexadecimal integer and fraction:

   $B6.6_{16} = (B6.6 \text{ X } 16^0)_{16}$

4. Shift the radix point:

   $(B6.6 \text{ X } 16^0)_{16} = (.B66 \text{ X } 16^2)_{16}$

5. Add 64, (X'40'), to the exponent

   $40_{16} + 2_{16} = 42_{16}$

6. Convert the exponent field and fraction to binary allowing 1 bit for the sign, 7 bits for the exponent field, and 24 or 56 bits for the fraction.

   42B66 = 0100 0010 1011 0110 0110 0000 0000 0000

CONDITION CODE

Following floating point operations, including load, the Condition Code indicates the result of the operation.

FLOATING POINT INSTRUCTION FORMATS

The Floating Point instructions use the Register to Register (RR), and the Register and Indexed Storage (RX) instruction formats. In all of the RR formats, except for Fix and Float, the R1 and the R2 fields specify one of the floating point registers. There are eight single precision floating point registers, and 8 double precision floating point registers numbered 0, 2, 4, 6, 8, 10, 12, and 14. Except FXR and FXDR instructions, the Rl field always specifies a floating point register.

FLOATING POINT INSTRUCTIONS

The floating point arithmetic operations, excluding loads and stores, require normalized operands to ensure correct results. If the operands are not normalized, the results of these operations are undefined. Floating point results are normalized. The Floating Point Load instruction normalizes floating point data extracted from memory.

The single precision floating point instructions described in this section are:

LE	Load Floating Point	CE	Compare Floating Point
LER	Load Floating Point Register	CER	Compare Floating Point Register
LME	Load Floating Point Multiple	ME	Multiply Floating Point
STE	Store Floating Point	MER	Multiply Floating Point Register
STME	Store Floating Point Multiple	DE	Divide Floating Point
AE	Add Floating Point	DER	Divide Floating Point Register
AER	Add Floating Point Register	FXR	Fix Register
SE	Subtract Floating Point	FLR	Float Register
SER	Subtract Floating Point Register		

The double precision floating point instructions described in this section are:

LD	Load DPFP	CD	Compare DPFP
LDR	Load Register DPFP	CDR	Compare Register DPFP
LMD	Load Multiple DPFP	MD	Multiply DPFP
STD	Store DPFP	MDR	Multiply Register DPFP
STMD	Store Multiple DPFP	DD	Divide DPFP
AD	Add DPFP	DDR	Divide register DPFP
ADR	Add Register DPFP	FXDR	Fix Register DPFP
SD	Subtract DPFP	FLDR	Float Register DPFP
SDR	Subtract Register DPFP		

Load Floating Point (LE)
Load Floating Point Register (LER)

Assembler Notation		Op-Code	Format
LE	R1,D2 (X2)	68	RX1,RX2
LE	R1,A2 (FX2,SX2)	68	RX3
LER	R1,R2	28	RR

Operation

The floating point second operand is normalized, if necessary, and placed in the floating point register specified by R1.

Condition Code

C	V	G	L	
0	0	0	0	Floating point value is ZERO
0	0	0	1	Floating point value is less than ZERO
0	0	1	0	Floating point value is greater than ZERO
0	1	0	0	Exponent underflow

Programming Note

If the fraction is zero, the result is forced to X'0000 0000'

Normalization may produce exponent underflow. In this event, the result is forced to zero, X'0000 0000', the V flag in the Condition Code is set, the G and L flags are reset and, if enabled by Bit 19 of the current PSW, the arithmetic fault interrupt is taken.

In the RX formats, the second operand must be located on a fullword boundary.

Example: LE

This example normalizes the fullword at memory location LOC and places it in Floating Point Register 8.

Floating Point Register 8	= undefined
LOC	= X'4200 1000'

Assembler Notation	Comments
LE REG8, LOC	Normalize contents of LOC

Result of LE Instruction

(Floating Point Register 8)	=	4010 0000
(LOC)	=	unchanged by this instruction
Condition Code	=	0010

INSTRUCTION

Load Floating Point Multiple (LME)

Assembler Notation		Op-Code	Format
LME	R1, D2 (X2)	72	RX1, RX2
LME	R1, A2 (FX2, SX2)	72	RX3

Operation

Successive floating point registers, starting with the register specified by R1, are loaded from successive memory locations starting with the address of the second operand. The process stops when Floating Point Register 14 has been loaded.

Condition Code

Unchanged

Programming Note

Values loaded into the floating point registers are not normalized first.

The second operand must be located on a fullword boundary.

INSTRUCTION

Store Floating Point (STE)

Assembler Notation		Op-Code	Format
STE	R1,D2 (X2)	60	RX1,RX2
STE	R1,A2 (FX2,SX2)	60	RX3

Operation

The floating point first operand, contained in the floating point register specified by R1, is placed in the memory location specified by the second operand address. The first operand is unchanged.

Condition Code

Unchanged

Programming Note

The second operand must be located on a fullword boundary.

INSTRUCTION

Store Floating Point Multiple (STME)

Assembler Notation		Op-Code	Format
STME	R1, D2 (X2)	71	RX1, RX2
STME	R1, A2 (FX2, SX2)	71	RX3

Operation

The contents of successive floating point registers, starting with the register specified by R1, are stored in successive memory locations, starting with the address of the second operand. The operation stops when the contents of Floating Point Register 14 have been stored.

Condition Code

Unchanged

Programming Note

The second operand must be located on a fullword boundary.

INSTRUCTIONS

Add Floating Point (AE)
Add Floating Point Register (AER)

Assembler Notation		Op-Code	Format
AE	R1, D2 (X2)	6A	RX1, RX2
AE	R1, A2 (FX2, SX2)	6A	RX3
AER	R1, R2	2A	RR

Operation

The exponents of the two operands are compared. If the exponents differ, the fraction with the smaller exponent is shifted right hexadecimally (four bits at a time), and its exponent is incremented by one for each hexadecimal shift until the two exponents are equal. The hexadecimal digits (of four bits each) are shifted through the guard digit. The guard digit contains the last shifted hexadecimal digit. If no shift occurs it is zero. The fractions are then added algebraically.

If the addition of fractions produces a carry, the exponent of the result is incremented by one and the fraction of the result is shifted right one hexadecimal digit. The carry bit is shifted back into the most significant hexadecimal digit of the fraction, producing a normalized result. This result replaces the contents of the register specified by R1.

If the addition of fractions does not produce a carry, the result is normalized, if necessary, and replaces the contents of the register specified by R1.

Condition Code

C	V	G	L	
X	0	0	0	Floating point result is ZERO
X	0	0	1	Floating point result is less than ZERO
X	0	1	0	Floating point result is greater than ZERO
X	1	0	1	Exponent overflow, Result is negative
X	1	1	0	Exponent overflow, Result is positive
X	1	0	0	Exponent underflow

Programming Note

When the addition of the fractions produces a carry, incrementing the exponent of the result by one may produce exponent overflow. In this case, the result is forced to the maximum value, +X'7FFF FFFF', the V flag, along with the G or L flag is set in the Condition Code and, if enabled by Bit 19 of the current PSW, the arithmetic fault interrupt is taken.

Normalization of the result may produce exponent underflow. In this case, the result is forced to zero, X'0000 0000'. The V flag is set in the Condition Code. The G and the L flags are always reset, and if enabled by Bit-19 of the current PSW, the arithmetic fault interrupt is taken.

If the guard digit is 0:7, the result is not rounded. If the guard digit is 8:F, the result is rounded by adding 1 to the fraction of the result unless rounding produces a carry into the exponent field.

In the RX formats, the second operand must be located on a fullword boundary.

Example: AE

This example adds the contents of LOC to the contents of the Floating Point Register 8 and places the answer in Floating Point Register 8.

Floating Point Register 8 contains X'7EFF FFFF'
LOC contains X'7EFF FFFF'

Assembler Notation	Comments
AE REG8, LOC	ADD (REG 8) to (LOC)

Result of AE Instruction

(Floating Point Register 8)	=	7F1F FFFF
(LOC)	=	unchanged by this instruction
Condition Code	=	0010

INSTRUCTIONS

Subtract Floating Point (SE)
Subtract Floating Point Register (SER)

Assembler Notation		Op-Code	Format
SE	R1,D2 (X2)	6B	RX1,RX2
SE	R1,A2 (FX2,SX2)	6B	RX3
SER	R1,R2	2B	RR

Operation

The exponents of the two operands are compared. If the exponents differ, the fraction with the smaller exponent is shifted right hexadecimally (four bits at a time), and its exponent is incremented by one for each hexadecimal shift until the two exponents are equal. The hexadecimal digits (of four bits each) are shifted through the guard digit. The guard digit contains the last shifted hexadecimal digit. If no shift occurs it is zero. The second operand fraction is then subtracted algebraically from the first operand fraction.

If the subtraction of fractions produces a carry, the exponent of the result is incremented by one and the fraction of the result is shifted right one hexadecimal digit. The carry bit is shifted back into the most significant hexadecimal digit of the fraction, producing a normalized result. This result replaces the contents of the register specified by R1.

If the subtraction of fractions does not produce a carry, the result is normalized. The normalized result replaces the contents of the register specified by R1.

Condition Code

C	V	G	L	
X	0	0	0	Floating point result is ZERO
X	0	0	1	Floating point result is less than ZERO
X	0	1	0	Floating point result is greater than ZERO
X	1	0	1	Exponent overflow, Result is negative
X	1	1	0	Exponent overflow, Result is positive
X	1	0	0	Exponent underflow

Programming Note

When the subtraction of the fractions produces a carry, incrementing the exponent of the result by one may produce exponent overflow. In this case, the result is forced to the maximum value, ±X'7FFF FFFF', the V flag, along with the G or L flag is set in the Condition Code and, if enabled by Bit-19 of the current PSW, the arithmetic fault interrupt is taken.

Normalization of the result may produce exponent underflow. In this case, the result is forced to zero, X'0000 0000'. The V flag is set in the Condition Code. The G and the L flags are always reset and, if enabled by Bit-19 of the current PSW, the arithmetic fault interrupt is taken.

The shifted hexadecimal digits (if any) participate in subtraction and produce a guard digit. If the guard digit is 0:7, the result is not rounded. If the guard digit is 8:F, the result is rounded by adding 1 to the fraction of the result unless rounding produces a carry into the exponent field.

In the RX formats, the second operand must be located on a fullword boundary.

Example: SE

This example subtracts the contents of LOC from the contents of Floating Point Register 8 and places the result in Floating Point Register 8.

Floating Point Register 8	contains X'7FEF FFFF'
LOC	contains X'7A10 0000'

Assembler Notation	Comments
SE REG8, LOC	Subtract (LOC) from REG8

Result of SE Instruction

(Floating Point Register 8)	=	7FEF FFFE
(LOC)	=	unchanged by this instruction
Condition Code	=	0010

INSTRUCTIONS

Compare Floating Point (CE)
Compare Floating Point Register (CER)

Assembler Notation		Op-Code	Format
CE	R1,D2 (X2)	69	RX1,RX2
CE	R1,D2 (FX2,SX2)	69	RX3
CER	R1,R2	29	RR

Operation

The first operand is compared to the second operand. Comparision is algebraic, taking into account the sign, fraction, and exponent of each number. The result is indicated by the Condition Code setting. Neither operand is changed.

Condition Code

C	V	G	L	
0	X	0	0	First operand is equal to second operand
1	X	0	1	First operand is less than second operand
0	X	1	0	First operand is greater than second operand

Programming Note

The state of the V flag is undefined.

In the RX formats, the second operand must be located on a fullword boundary.

Multiply Floating Point (ME)
Multiply Floating Point Register (MER)

Assembler Notation		Op-Code	Format
ME	R1, D2 (X2)	6C	RX1, RX2
ME	R1, A2 (FX2, SX2)	6C	RX3
MER	R1, R2	2C	RR

Operation

The exponents of each operand, as derived from the excess 64 notation used in floating point representation, are added to produce the exponent of the result. This exponent is converted back to excess 64 notation. The fractions are then multiplied.

If the result is zero, the entire floating point value is forced to zero, X'0000 0000'. If the product is not zero, the result is normalized. The sign of the result is determined by the rules of algebra. The result replaces the contents of the register specified by R1.

Condition Code

C	V	G	L	
X	0	0	0	Floating point result is ZERO
X	0	0	1	Floating point result is less than ZERO
X	0	1	0	Floating point result is greater than ZERO
X	1	0	1	Exponent overflow, Result is negative
X	1	1	0	Exponent overflow, Result is positive
X	1	0	0	Exponent underflow

Programming Note

The addition of exponents may produce exponent overflow. In this case, the result is forced to the maximum value, ±X'7FFF FFFF'. The V flag in the Condition Code is set, along with either the G or the L flag, depending on the sign of the result. An arithmetic fault interrupt is taken, if enabled by Bit-19 of the current PSW.

The addition of exponents or the normalization process can produce exponent underflow. In this case, the result is forced to zero, X'0000 0000'. The V flag in the Condition Code is set. The G and L flags are reset, and if enabled by Bit-19 of the current PSW, the arithmetic fault interrupt is taken.

Multiplication of two 6-hexadecimal digit fractions effectively produces a result of 6-hexadecimal digits and a guard digit. If the guard digit is 0:7, the result is not rounded. If the guard digit is 8:F, the result is rounded by adding 1 to the fraction of the result, unless rounding produces a carry into the exponent field.

In the RX formats, the second operand must be located on a fullword boundary.

Example: ME

This example multiplies the contents of LOC by the contents of the Floating Point Register 8 and places the result in Floating Pointer Register 8.

Floating Point Register 8	contains X'5FFF FFFF'
LOC	contains X'60FF FFFF'

Assembler Notation	Comments
ME REG8, LOC	Multiply (REG 8) by (LOC)

Result of ME Instruction

(Floating Point Register 8)	=	7FFF FFFE
(LOC)	=	unchanged by this instruction
Condition Code	=	0010

Divide Floating Point (DE)
Divide Floating Point Register (DER)

Assembler Notation		Op-Code	Format
DE	R1,D2 (X2)	6D	RX1,RX2
DE	R1,A2 (FX2,SX2)	6D	RX3
DER	R1,R2	2D	RR

Operation

The exponents of each operand, as derived from the excess of 64 notation used in floating point representation, are subtracted to produce the exponent of the result. This exponent is converted back to excess 64 notation.

The first operand fraction is then divided by the second operand fraction. Division continues until the quotient is normalized, adjusting the exponent for each additional division required. No remainder is returned. The sign of the quotient is determined by the rules of algebra. The quotient replaces the contents of the register specified by R1.

Condition Code

C	V	G	L	
0	0	0	0	Floating point result is ZERO
0	0	0	1	Floating point result is less than ZERO
0	0	1	0	Floating point result is greater than ZERO
0	1	0	1	Exponent overflow, Result is negative
0	1	1	0	Exponent overflow, Result is positive
0	1	0	0	Exponent underflow
1	1	0	0	Divisor equal to zero

Programming Note

Before starting the divide operation, the divisor is checked. If it is equal to zero, the operation is aborted. Neither operand is changed. The C and the V flags of the Condition Code are set. The G and L flags are reset. If enabled by Bit-19 of the current PSW, the arithmetic fault interrupt is taken.

The subtraction of exponents may produce exponent overflow. In this case, the result is forced to the maximum value, ±X'7FFF FFFF'. The V flag in the Condition Code is set, along with either the G or the L flag, depending on the sign of the result. An arithmetic fault interrupt is taken, if enabled by Bit-19 of the current PSW.

The subtraction of exponents or the division process can produce exponent underflow. In this case, the result is forced to zero, X'0000 0000'. The V flag in the Condition Code is set. The G and L flags are always reset, and if enabled by Bit-19 of the current PSW, the arithmetic fault interrupt is taken.

The 6-hexadecimal digit first operand fraction is divided by the 6-hexadecimal digit second operand effectively producing the 6-hexadecimal digit quotient along with a guard digit. If the guard digit is 0:7, the quotient is not rounded. If the guard digit is 8:F, the quotient is rounded by adding 1 to the fraction of the result unless rounding produces a carry into the exponent field.

In the RX formats, the second operand must be located on a fullword boundary.

Example: DE

This example divides the contents of Floating Point Register 4 by the contents of memory location LOC and places the result in Floating Pointer Register 4.

Floating Point Register 4	contains X'44FF FFFF'	= Dividend
LOC	contains X'0611 1111'	= Divisor

Assembler Notation	Comments
DE REG4, LOC	Divide (LOC) into (REG 4)

Result of DE Instruction

(Floating Point Register 4)	=	7FF0 0000
(LOC)	=	unchanged by this instruction
Condition Code	=	0010

INSTRUCTION

Fix Register (FXR)

Assembler Notation		Op-Code	Format
FXR	R1,R2	2E	RR

Operation

R1 specifies one of the general purpose registers. R2 specifies one of the floating point registers. The floating point number contained in the floating point register is converted to a two's complement notation integer value by shifting and truncating. The result is stored in the register specified by R1.

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO or underflow
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO
X	1	0	1	Overflow, Result is negative
X	1	1	0	Overflow, Result is positive

Programming Note

The range of floating point magnitudes M that produces a non-zero integral result is:

$\pm$X'4880 0000' > M $\geq$ $\pm$X'4110 0000'

Floating point magnitudes greater than +X'487F FFFF' cause overflow. The result is forced to X'7FFF FFFF' if positive or to X'8000 0001' if negative. The V flag is set in the Condition Code along with either the G or L Flag, depending on the sign of the result.

Floating point magnitudes less than +X'4110 0000' cause underflow and the result is forced to zero.

In the event of overflow or underflow, the Arithmetic Fault Interrupt is not taken, even if enabled in the current PSW.

Example: FXR

This example converts the contents of the Floating Point Register 8 to a fixed point number and places it in Register 3.

Floating Point Register 8 contains X'46FF FF00'
Register 3 contains undefined

Assembler Notation	Comments
FXR REG3, REG8	Convert (REG 8) to fixed point

Result of FXR Instruction

(REG3)	=	00FFFF00
(Floating Point Register 8)	=	unchanged by this instruction
Condition Code	=	0010

INSTRUCTION

Float Register (FLR)

Assembler Notation		Op-Code	Format
FLR	R1,R2	2F	RR

Operation

R1 specifies one of the floating point registers. R2 specifies one of the general purpose registers. The integer value contained in the register specified by R2 is converted to a floating point number and stored in the floating point register specified by R1.

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO

Programming Note

The full range of fixed point integer values may be converted to floating point. The fixed point value X'7FFF FFFF', the largest positive integer, converts to a floating point value of X'487F FFFF'. The fixed point value X'8000 0000', the most negative integer, converts to a floating point value of X'C880 0000'. The result in Rl is normalized.

Example: FLR

This example converts the Fixed point contents of Register 4 to a Floating Point number and places it into Floating Point Register 8.

Register 4	contains X'7FFF FFF0'
Floating Point Register 8	contents undefined

Assembler Notation	Comments
FLR REG8, REG4	Convert REG4 to Floating Point

Result of FLR Instruction

(Floating Point Register 8)	=	487FFFFF
(REG4)	=	unchanged by this instruction
Condition Code	=	0010

INSTRUCTIONS

Load Double Precision Floating Point (LD)
Load Register Double Precision Floating Point (LDR)

Assembler Notation	Op-Code	Format
LD R1,D2(X2)	78	RX1,RX2
LD R1,A2,(FX2,SX2)	78	RX3
LDR R1,R2	38	RR

Operation

The floating point second operand is normalized, if necessary, and placed in the double precision floating point register specified by R1.

Condition Code

C	V	G	L	
0	0	0	0	Double precision value is ZERO
0	0	0	1	Double precision value is less than ZERO
0	0	1	0	Double precision value is greater than ZERO
0	1	0	0	Exponent underflow

Programming Note

If the fraction is zero, the result is forced to X'0000 0000 0000 0000'.

Normalization may produce exponent underflow. In this event, the result is forced to X'0000 0000 0000 0000', the V flag in the Condition Code is set, the G and L flags are reset and, if enabled by Bit-19 of the current PSW, the arithmetic fault interrupt is taken.

In the RX formats, the second operand must be located on a double word boundary.

INSTRUCTION

Load Multiple Double Precision Floating Point (LMD)

Assembler Notation	Op-Code	Format
LMD R1, D2(X2)	7F	RX1, RX2
LMD R1, A2(FX2, SX2)	7F	RX3

Operation

Successive double-precision floating point registers, starting with the register specified by R1, are loaded from successive memory locations starting with the address of the second operand. The process stops when Double Precision Floating Point Register 14 has been loaded.

Condition Code

Unchanged

Programming Note

Values loaded into the double precision floating point registers are not normalized first.

The second operand must be located on a double word boundary.

INSTRUCTION

Store Double Precision Floating Point (STD)

Assembler Notation	Op-Code	Format
STD R1, D2, (X2)	70	RX1, RX2
STD R1, A2(FX2, SX2)	70	RX3

Operation

The floating point first operand, contained in the double precision floating point register specified by R1 is placed in the memory location specified by the second operand address. The first operand is unchanged.

Condition Code

Unchanged.

Programming Note

The second operand must be located on a double word boundary.

INSTRUCTION

Store Multiple Double Precision Floating Point (STMD)

Assembler Notation	Op-Code	Format
STMD R1, D2(X2)	7E	RX1, RX2
STMD R1, A2 (FX2, SX2)	7E	RX3

Operation

The contents of successive double precision floating point registers, starting with the register specified by R1, are stored in successive memory locations, starting with the address of the second operand. The operation stops when the contents of Double Precision Floating Point Register 14 have been stored.

Condition Code

Unchanged

Programming Note

The second operand must be located on a double word boundary.

INSTRUCTIONS

Add Double Precision Floating Point (AD)
Add Register Double Precision Floating Point (ADR)

Assembler Notation	Op-Code	Format
AD R1, D2(X2)	7A	RX1, RX2
AD R1, A2(FX2, SX2)	7A	RX3
ADR R1, R2	3A	RR

Operation

The exponents of the two operands are compared. If the exponents differ the fraction with the smaller exponent is shifted right hexadecimally (four bits at a time), and its exponent is incremented by one for each hexadecimal shift until the two exponents are equal. The fractions are then added algebraically.

If the addition of fractions produces a carry, the exponent of the result is incremented by one and the fraction of the result is shifted right one hexadecimal position. The carry bit is shifted back into the most significant hexadecimal digit of the fraction, producing a normalized result. This result replaces the contents of the double precision floating point register specified by R1.

If the addition of fractions does not produce a carry, the result is normalized, if necessary, and placed in the double precision floating point register specified by R1.

Condition Code

C	V	G	L	
X	0	0	0	Double Precision Result is ZERO
X	0	0	1	Double Precision Result is less than ZERO
X	0	1	0	Double Precision Result is greater than ZERO
X	1	0	1	Exponent Overflow, Result is negative
X	1	1	0	Exponent Overflow, Result is positive
X	1	0	0	Exponent Underflow

Programming Note

When the addition of fractions produces a carry, incrementing the exponent of the result by one may produce exponent overflow. In this case, the result is forced to the maximum value, ±X'7FFF FFFF FFFF FFFF', the V flag, along with the G or L flag is set in the Condition Code and, if enabled by Bit-19 of the current PSW, the arithmetic fault interrupt is taken.

Normalization of the result may produce exponent underflow. In this case, the result is forced to zero, X'0000 0000 0000 0000'. The V flag is set in the Condition Code, and the G and L flags are reset, and if enabled by Bit-19 of the current PSW, the arithmetic fault interrupt is taken.

In the RX formats, the second operand must be located on a double word boundary.

INSTRUCTIONS

Subtract Double Precision Floating Point (SD)
Subtract Register Double Precision Floating Point (SDR)

Assembler Notation	Op-Code	Format
SD R1, D2(X2)	7B	RX1, RX2
SD R1, A2(FX2, SX2)	7B	RX3
SDR R1, R2	3B	RR

Operation

The exponents of the two operands are compared. If the exponents differ, the fraction with the smaller exponent is shifted right hexadecimally (four bits at a time), and its exponent is incremented by one for each hexadecimal shift until the two exponents are equal. The second operand fraction is then subtracted algebraically from the first operand fraction.

If the subtraction of fractions produces a carry, the exponent of the result is incremented by one and the fraction of the result is shifted right one hexadecimal position. The carry bit is shifted back into the most significant hexadecimal digit of the fraction producing a normalized result. This result replaces the contents of the double precision floating point register specified by R1.

Condition Code

C	V	G	L	
X	0	0	0	Double Precision Result is ZERO
X	0	0	1	Double Precision Result is less than ZERO
X	0	1	0	Double Precision Result is greater than ZERO
X	1	1	0	Exponent Overflow, Result is positive
X	1	0	1	Exponent Overflow, Result is negative
X	1	0	0	Exponent Underflow

Programming Note

When the subtraction of fractions produces a carry, incrementing the exponent of the result by one may produce exponent overflow. In this case, the result is forced to the maximum value, ± X'7FFF FFFF FFFF FFFF', the V flag, along with the G or L flag is set in the Condition Code, and if enabled by Bit-19 of the current PSW, the arithmetic fault interrupt is taken.

Normalization of the result may produce exponent underflow. In this case, the result is forced to zero, X'0000 0000 0000 0000'. The V flag is set in the Condition Code, the G and L flags are reset, and if enabled by Bit-19 of the current PSW, the arithmetic fault interrupt is taken.

In the RX formats, the second operand must be located on a double word boundary.

INSTRUCTIONS

Compare Double Precision Floating Point (CD)
Compare Register Double Precision Floating Point (CDR)

Assembler Notation	Op-Code	Format
CD R1,D2(X2)	79	RX1,RX2
CD R1,A2(FX2,SX2)	79	RX3
CDR R1,R2	39	RR

Operation

The first operand is compared to the second operand. Comparison is algebraic, taking into account the sign, exponent and fraction of each number. The result is indicated by the Condition Code setting. Neither operand is changed.

Condition Code

C	V	G	L	
0	X	0	0	First operand is equal to second operand
1	X	0	1	First operand is less than second operand
0	X	1	0	First operand is greater than second operand

Programming Note

The state of the overflow flag is undefined.

In the RX formats, the second operand must be located on a double word boundary.

Multiply Double Precision Floating Point (MD)
Multiply Register Double Precision Floating Point (MDR)

Assembler Notation	Op-Code	Format
MD R1, D2(X2)	7C	RX1, RX2
MD R1, A2(FX2, SX2)	7C	RX3
MDR R1, R2	3C	RR

Operation

The exponents of the two operands, as derived from the excess 64 notation used in floating point representation, are added to produce the exponent of the result. This exponent is converted back to excess 64 notation. The fractions are then multiplied.

If the product is zero, the entire double precision value is forced to zero, X'0000 0000 0000 0000'. If the product is not zero, the result is normalized if necessary. After normalization, the product is rounded. The sign of the result is determined by the rules of algebra. The result replaces the contents of the double precision floating point register specified by R1.

Condition Code

C	V	G	L	
X	0	0	0	Double precision result is ZERO
X	0	0	1	Double precision result is less than ZERO
X	0	1	0	Double precision result is greater than ZERO
X	1	1	0	Exponent overflow, Result is positive
X	1	0	1	Exponent overflow, Result is negative
X	1	0	0	Exponent underflow

Programming Note

The addition of exponents may produce exponent overflow. In this case, the result is forced to the maximum value, ±X'7FFF FFFF FFFF FFFF'. The V flag in the Condition Code is set, along with either the G or L flag, depending on the sign of the result. An arithmetic fault interrupt is taken, if enabled by Bit-19 of the current PSW.

The addition of exponents or the normalization process can produce exponent underflow. In this case, the result is forced to zero, X'0000 0000 0000 0000'. The V flag in the Condition Code is set, the G and L flags are reset, and if enabled by Bit 19 of the current PSW, the arithmetic fault interrupt is taken.

In the RX formats, the second operand must be located on a double word boundary.

Divide Double Precision Floating Point (DD)
Divide Register Double Precision Floating Point (DDR)

Assembler Notation	Op-Code	Format
DD R1, D2 (X2)	7D	RX1, RX2
DD R1, A2 (FX2SX2)	7D	RX3
DDR R1, R2	3D	RR

Operation

The exponents of the two operands, as derived from the excess 64 notations used in floating point representation, are subtracted to produce the exponent of the result. This exponent is converted back to excess 64 notation.

The second operand fraction is then divided into the first operand fraction. Division continues until the quotient is normalized, adjusting the exponent for each additional division required.

No remainder is returned. The sign of the result is determined by the rules of algebra. The quotient replaces the contents of the double precision floating point register specified by R1.

Condition Code

C	V	G	L	
0	0	0	0	Double precision result is ZERO
0	0	0	1	Double precision result is less than ZERO
0	0	1	0	Double precision result is greater than ZERO
0	1	0	1	Exponent overflow, Result is negative
0	1	1	0	Exponent overflow, Result is positive
0	1	0	0	Exponent underflow
1	1	0	0	Divisor is zero

Programming Note

Before starting the divide operation, the divisor is checked. If it is equal to zero, the operation is aborted. Neither operand is changed. The C and V flags in the Condition Code are set, the G and L flags are reset, and if enabled by Bit 19 of the current PSW, the arithmetic fault interrupt is taken.

The subtraction of exponents may produce exponent overflow. In this case, the result is forced to the maximum value, ±X'7FFF FFFF FFFF FFFF'. The V flag in the Condition Code is set, along with either the G or L flag, depending on the sign of the result. An arithmetic fault interrupt is taken, if enabled by Bit-19 of the current PSW.

The subtraction of exponents or the division process may produce exponent underflow. In this case, the result is forced to zero, X'0000 0000 0000 0000'. The V flag in the Condition Code is set, the G and L flags are reset, and if enabled by Bit-19 of the current PSW, the arithmetic fault interrupt is taken.

In the RX formats, the second operand must be located on a double word boundary.

INSTRUCTION

Fix Register Double Precision (FXDR)

Assembler Notation	Op-Code	Format
FXDR R1, R2	3E	RR

Operation

R1 specifies one of the general purpose registers. R2 specifies one of the double precision floating point registers. The floating point number contained in the floating point register is converted to an integer value by truncating. The result is placed in the general register specified by R1.

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO or underflow
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO
X	1	0	1	Overflow, Result is negative
X	1	1	0	Overflow, Result is positive

Programming Note

The range of the floating point magnitude M that produces a non-zero integral result is,
± X'4880 0000 0000 0000' > M ≥ ± X'4110 0000 0000 0000'

Double precision floating point magnitudes greater than +X'487F FFFF FFFF FFFF' cause overflow. The result is forced to X'7FFF FFFF' if positive or to X'8000 0001' if negative. The V flag is set in the Condition Code along with either G or L flag, depending on the sign of the result.

Double Precision floating point magnitudes less than +X'4110 0000 0000 0000' cause underflow. The result is forced to zero and the Condition Code is set to zero.

In the event of overflow or underflow, the Arithmetic Fault Interrupt is not taken even if enabled in the current PSW.

INSTRUCTION

Float Register Double Precision (FLDR)

Assembler Notation	Op-Code	Format
FLDR R1, R2	3F	RR

Operation

R1 specifies one of the double precision floating point registers. R2 specifies one of the general purpose registers. The integer value contained in the register specified by R2 is converted to a floating point number and placed in the double precision floating point register specified by R1.

Condition Code

C	V	G	L	
X	0	0	0	Result is ZERO
X	0	0	1	Result is less than ZERO
X	0	1	0	Result is greater than ZERO

Programming Note

The full range of fixed point integer values may be converted to double precision floating point. The fixed point value X'7FFF FFFF', the largest positive integer, converts to a double precision floating point value of X'487F FFFF FF00 0000'. The fixed point value X'8000 0000', the most negative integer, converts to a double precision floating point value of X'C880 0000 0000 0000'.

The result in R1 is normalized.

CHAPTER 6
STATUS SWITCHING AND INTERRUPTS

At any given time, the Processor may be in either the Stop mode or the Run mode. In the Stop mode, the normal execution of instructions is suspended. The Processor is under control of the operator who can, through the display console:

Examine any memory location

Change any memory location

Examine the contents of any general register

Examine and modify the current PSW

Execute instructions singly

The transition from the Stop mode to the Run mode requires operator intervention at the display console, or the occurrence of an interrupt (if enabled by the current PSW).

Once the Processor has been put in the Run mode, the current PSW controls the operation of the Processor. By changing the contents of the current PSW, a running program can:

Put the Processor in the Wait state

Enable or disable various interrupts

Switch between supervisor and protect modes

Vary the normal sequential execution of instructions

PROGRAM STATUS WORD

The Program Status Word is a 64-bit double word. (See Figure 6-1.)

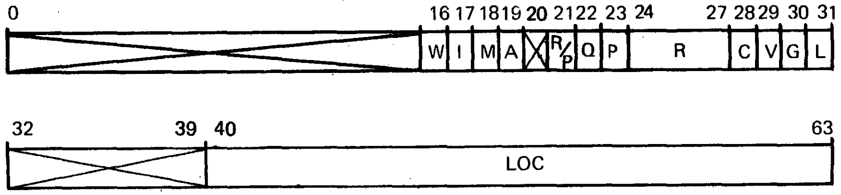


Figure 6-1. Program Status Word

Bits 0:15 of the PSW are not currently used, and must be zero. Bits 16:27 are reserved for status definition and interrupt masks. Bit 20 is not currently used, and must be zero. Bits 28:31 are reserved for the Condition Code. Bits 32:39 are not used, and must be zero. Bits 40:63 are reserved for the Location Counter. The status and interrupt bits are interpreted as follows:

Bit 16 (W)	Wait state
Bit 17 (I)	Immediate interrupt/Auto Driver Channel enable
Bit 18 (M)	Machine malfunction interrupt enable
Bit 19 (A)	Arithmetic fault interrupt enable
Bit 21 (RP)	Relocation/protection enable
Bit 22 (Q)	System queue service interrupt enable
Bit 23 (P)	Protect mode
Bits 24:27 (R)	Register set selection

The current PSW is contained in a hardware register within the Processor. Status switching results when the current PSW, or at least the first half (Bits 0:31) of the current PSW, is replaced. The occurrence of an interrupt or the execution of a Status Switching instruction can cause the replacement of the current PSW.

Wait State

When Bit 16 of the current PSW is set, the Processor is in the Wait state. In this state, program execution is halted. However, the Processor is still responsive to machine malfunction and immediate interrupts, if they are enabled. If the Processor is put in the Wait state with these interrupts disabled, only operator intervention from the Display console can force the Processor out of the Wait state.

Protect Mode

When Bit-23 of the current PSW is set, the Processor is in the protect mode. A program running in this mode is not allowed to execute Privileged instructions. (Privileged instructions include all I/O instructions and most of the Status Switching instructions. See Appendix 1.) A privileged instruction is treated as an illegal instruction when the Processor is in the protect mode. If Bit-23 of the current PSW is reset, the Processor is in the Supervisor mode. Programs running in this mode may execute any legal instruction.

Register Set Selection

Model 7/32 has two register sets, numbered 0 and 15. Bits 24:27 of the current PSW control register set selection. If Bits 24:27 are all zeroes, register set 0 is selected. If Bits 24:27 are all ones, register set 15 is selected.

NOTE

In Model 7/32, Bits 24, 25, 26 of the current PSW have no effect on selection of register sets. Consequently, specifying an even numbered register set causes register set 0 to be selected whereas specifying odd numbered register set causes register set 15 to be selected.

INTERRUPT SYSTEM

The interrupt system of the Processor provides rapid response to external and internal events that require service by special software routines. In the interrupt response procedure, the Processor preserves its current state and transfers control to the required interrupt handler. This software routine may optionally restore the previous state of the Processor upon completion of the service. (See Table 6-1 and Figure 6-2.)

Some interrupts are controlled by bits in the current Program Status Word, that is, they can be enabled or disabled by setting or resetting a bit in the PSW. Other interrupts are not controlled by PSW bits, and are always enabled. The following is a list of Processor interrupts and their controlling PSW bits, if any:

Interrupt	PSW Bit
Immediate, Auto Driver Channel	17
Console	17
Machine Malfunction	18
Arithmetic Fault	19
Relocation/Protection	21
System Queue Service	22
Protect Mode Violation	23
Supervisor Call	none
Simulated	none
Illegal Instruction	none

Interrupts occur at various times during processing. The immediate, console, and machine malfunction interrupts occur between the execution of instructions or after completion of an auto driver channel operation. The relocation/protection interrupt occurs after the execution of an instruction. The system queue service, arithmetic fault, supervisor call, and simulated interrupts occur during the execution of instructions. The illegal instruction and protect mode violation interrupts occur before the execution of the improper instruction.

The interrupt procedure is based on the concepts of old, current, and new Program Status Words. The current PSW, contained in a hardware register, defines the operating state of the Processor. When this state must be changed, the current PSW becomes the old PSW. The new PSW becomes the current PSW. The current PSW now contains the operating status and the Location Counter for the interrupt service routine.

With one exception (the machine malfunction interrupt), when the current PSW bcomes the old PSW it is saved in a pair of registers of register set 0. The machine malfunction old PSW is stored in a reserved memory location. Again with one exception, when a new PSW becomes the current PSW, it is loaded from a reserved memory location. The exception is the immediate interrupt. On an immediate interrupt, the current status is forced to a predetermined value. The current Location Counter is loaded from the interrupt service pointer table.

The new Program Status Word for any interrupt should, if possible, disable interrupts of its own class.

TABLE 6-1. INTERRUPT SYSTEMS

INTERRUPT	TYPE	CONTROLLED BY PSW BIT (S)	CAN BE QUEUED	OLD PSW STORED IN	NEW PSW LOADED FROM	NOTES
ARITHMETIC FAULT	INTERNAL	19	NO	EXEC ~~REG. 14~~, 15	X'48 – 4F'	'C' FLAG SET IN NEW PSW IF FLOATING POINT ARITHMETIC FAULT
AUTO DRIVER CHANNEL	EXTERNAL	17	YES	REG. 0, 1	MICROPROGRAM (STATUS) CCB SUBROUTINE ADDRESS (LOC)	NEW PSW STATUS = Y'0000280X' (MACHINE MALFUNCTION INTERRUPT ENABLED) X IS THE CONDITION CODE ON TERMINATION (REG2) = INTERRUPTING DEVICE ADDRESS (REG3) = INTERRUPTING DEVICE STATUS (REG 4) = ADDRESS OF CHANNEL COMMAND BLOCK
CONSOLE	EXTERNAL	17	NO	(SEE IMMEDIATE INTERRUPT)		NEW PSW STATUS = Y'00002800'
ILLEGAL INSTRUCTION	INTERNAL	*	NO	REG. 14, 15	X'30-37'	CANNOT BE DISABLED
IMMEDIATE MUX BUS	EXTERNAL	17	YES	REG. 0, 1	MICROPROGRAM (STATUS) INTERRUPT SERVICE POINTER TABLE (LOC)	NEW PSW STATUS = Y'00002800' (MACHINE MALFUNCTION INTERRUPT ENABLED) (REG.2) = INTERRUPTING DEVICE ADDRESS (REG.3) = INTERRUPTING DEVICE STATUS
MACHINE MALFUNCTION	INTERNAL	18	YES	X'20 – 27'	X'38-3F'	CONDITION CODE SET TO INDICATE NATURE OF MALFUNCTION
MEMORY ACCESS CONTROLLER	INTERNAL	21	YES	REG. 14, 15	X'90-97'	MAC STATUS REGISTER INDICATES NATURE OF INTERRUPT
PRIVILEGED INSTRUCTION	INTERNAL	23	NO	REG. 14, 15	X'30-37'	TAKEN WHEN PRIVILEGED INSTRUCTION ATTEMPTED WHILE PROCESSOR IN PROTECT MODE
SIMULATED	INTERNAL	*	NO	(SEE IMMEDIATE OR AUTO DRIVER CHANNEL INTERRUPTS)		TAKEN WHEN 'SINT' INSTRUCTION EXECUTED IN NON-PROTECT MODE
SUPERVISOR CALL	INTERNAL	*	NO	REG. 14, 15	X'98-9B' (STATUS) SVC POINTER TABLE (LOC)	TAKEN WHEN 'SVC' INSTRUCTION EXECUTED (REG. 13) = ADDRESS OF SVC PARAMETER BLOCK
SYSTEM QUEUE	INTERNAL	22	YES	REG. 14, 15	X'88-8F'	TAKEN WHEN 'EPSR,' 'LPSW,' OR 'LPSWR' INSTRUCTIONS EXECUTED IF SYSTEM QUEUE NOT EMPTY. (REG. 13) = ADDRESS OF SYSTEM QUEUE

* This Interrupt is always enabled.

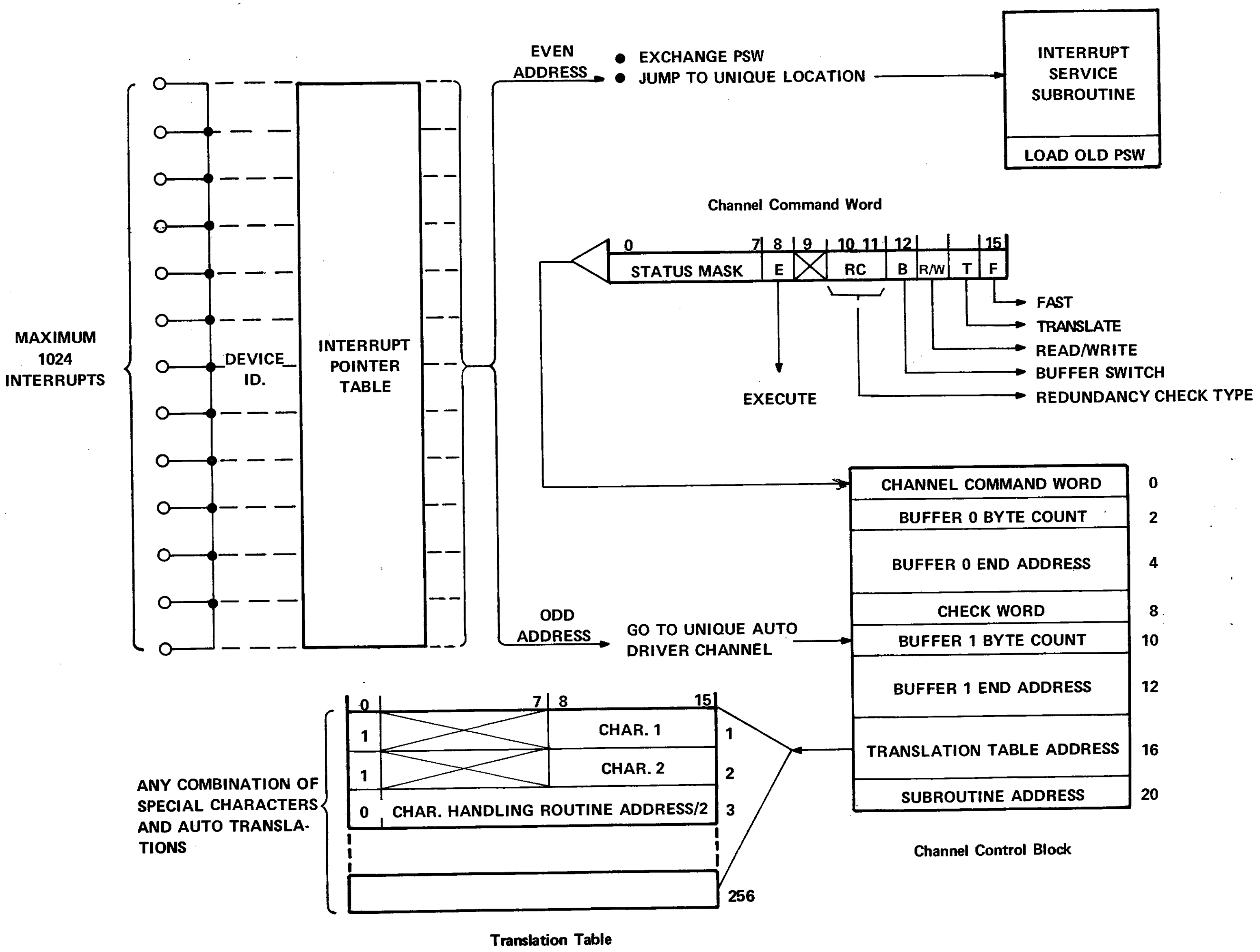


Figure 6-2. Interrupt Systems Block Diagram

Immediate Interrupt

The immediate interrupt is used for control of external devices. Through this mechanism, external devices can request and obtain Processor service.

When the Processor recognizes a request from a device, and Bit 17 of the current PSW is set, it:

1. Saves the current PSW in registers zero and one of the register set 0. (Bits 0:31 are saved in register zero; bits 32:63 are saved in register one.)
2. Loads the status portion of the current PSW with a value of X'00002800'.
3. Acknowledges the request and obtains the device number and status from the device. The device number is placed in register two. The status is placed in register three of the register set 0.
4. Adds two times the device number to X'0000D0' (the start of the interrupt service pointer table), to obtain the address within the table that corresponds to the interrupting device. For the immediate interrupt, the value in the table must be even. The value in the table becomes the current location counter.

In setting up the registers for the immediate interrupt service routine, the Processor loads the device number and status into the least significant bits of registers two and three. The most significant bits in these registers are forced to ZERO. Note that the new PSW disables immediate interrupt and specifies register set 0. The machine malfunction interrupt is enabled. Relocation and protection are disabled.

Console Interrupt

The console interrupt is a special case of the immediate interrupt. It also is controlled by Bit 17 of the current PSW. If Bit-17 is set, a console interrupt is generated by:

Depressing the Function key on the console, and then,

Depressing 0

The effect of the console interrupt is to cause an immediate interrupt, as described previously, from device number X'001'.

Simulated Interrupt

The Simulate Interrupt instruction simulates an immediate interrupt. When this instruction is executed, the Processor goes through the immediate interrupt procedure as if a request for service had been received from an external device. The current PSW is saved, and the current PSW loaded just as for the immediate interrupt. The device is addressed, and the status returned in Register 3. The address from the interrupt service pointer table is placed in Register 4. The state of Bit-17 has no effect on this interrupt. It is always enabled. The new register set is specified by the least significant 4 bits of the register specified by the R1 field of the instruction.

Machine Malfunction Interrupt

Bit-18 of the current PSW controls the machine malfunction interrupt. This interrupt occurs on a memory parity error, on the detection of primary power failure, and during the restart procedure after power has been restored. When a machine malfunction interrupt occurs, the current PSW is saved in memory location X'000020'. The new PSW from memory location X'000038' becomes the current PSW. The Condition Code of the new PSW as stored in memory must contain zeros. After the interrupt is taken, the state of the Condition Code indicates the specific cause of the interrupt.

Condition Code states are:

C	V	G	L	
0	0	0	0	Power Restore
0	0	0	1	Power failure
0	0	1	0	Memory malfunction (Parity Error on instruction fetch)
0	1	0	0	Memory malfunction (Parity error on data fetch)
1	1	0	0	Memory malfunction during Auto Driver Channel operation
1	0	0	1	Power failure during Auto Driver Channel operation

Power failure occurs when the primary power fail detector senses a low voltage, when the Initialize key (INT) of the Display console is depressed, or when the key operated POWER switch is turned to the OFF position. Following the PSW exchange, the software has approximately one millisecond to perform any necessary operations before the automatic shut down procedure takes over. During the automatic shut down procedure the Processor saves the current PSW at the memory location specified by the contents of location X'00084'; saves the 8 single-precision floating point registers, if equipped, in memory locations X'00000' through '0001F'; and it saves both sets of general registers, starting with register set 0, at the location specified by the contents of memory locaticn X'00086'. If the processor is equipped with double precision floating point, the double precision floating point registers are stored immediately following the General Register Save area.

When power returns, the Processor restores the PSW and the general registers and floating point registers from their save areas. If Bit 18 of the restored PSW is set, the Processor takes another machine malfunction interrupt, this time with no bits set in the Condition Code of the current PSW.

During Write operations to memory, with parity option, the Parity bit of each memory word is set to maintain odd parity. The Parity bit is recomputed on each memory read. If the computed bit is not equal to the bit read out of memory, the Processor takes a machine malfunction interrupt, setting the G flag to indicate the parity error.

If a machine malfunction interrupt condition arises during an auto driver channel operation, the PSW, current at the time the channel was activated, becomes the old machine malfunction PSW. Register 4 of the set, designated by the machine malfunction new PSW, contains the address of the Channel Command Block. The C flag of the current PSW is set along with either the L flag or the V flag to indicate either power failure or parity error.

Arithmetic Fault Interrupt·

Bit-19 of the current PSW controls the arithmetic fault interrupt. This interrupt, if enabled, can occur for any of the following reasons:

Fixed point division by zero
Fixed point quotient overflow
Floating point division by zero
Floating point overflow or underflow

When this interrupt occurs, the current PSW is saved in Registers 14 and 15 of the register set 0. The new PSW, from memory location X'000048', becomes the current PSW. All Condition Code bits in the new PSW as stored in memory must be zero. Before going to the interrupt service routine, the Processor sets the carry flag in the Condition Code if the interrupt is the result of a floating point operation. If the interrupt is the result of a fixed point operation, the carry flag is reset.

Any of the following conditions cause fixed point quotient overflow:

A halfword divide operation produces a result greater than 32,767 (X'7FFF').

A halfword divide operation produces a result less than -32,768 (X'8000').

A fullword divide operation produces a result greater than 2,147,483,647 (X'7FFF FFFF').

A fullword divide operation produces a result less than -2,147,483,648 (X'8000 0000').

When a fixed point division by zero or a fixed point quotient overflow occurs, the operand registers remain unchanged.

Floating point overflow occurs when, in a floating point operation, the value of the exponent exceeds +63. Floating point underflow occurs when, during the execution of a Floating Point instruction, the value of the exponent becomes less than -64. Following floating point overflow, the result is forced to plus or minus X'7FFF FFFF'. Following a floating point underflow, the result is forced to true zero, X'0000 0000'. After a floating point division by zero, the operand register remains unchanged.

After any arithmetic fault interrupt, the Location Counter of the old PSW contains the address of the instruction immediately following the one that caused the interrupt.

Relocation/Protection Interrupt

Bit-21 of the current PSW controls the relocation/protection interrupt. If this bit is set, and the currently running program violates any of the relocation and protection conditions available in the relocation and protection module, the Processor saves the current PSW in Registers 14 and 15 of the register set 0. The new PSW at memory location X'000090' becomes the current PSW.

System Queue Service Interrupt

Memory location X'000080' contains the address of the system queue. In the course of executing any of the following instructions:

Load Program Status Word
Load Program Status Word Register
Exchange Program Status Register

the Processor tests Bit-22 of the new status being loaded. If this bit is set, the Processor checks the state of the system queue. If there is an entry in the queue, the just loaded PSW becomes the old PSW. It is saved in Registers 14 and 15 of the register set 0. The address of the queue, taken from location X'000080', is placed in Register 13 of that set. The new PSW from location X'000088' becomes the current PSW.

Protect Mode Violation Interrupt

Bit-23 of the current PSW controls the execution of Privileged instructions. When this bit is set, the Processor is in the Protect mode. Programs running in the Protect mode are not allowed to execute Privileged instructions. Privileged instructions are:

All I/O instructions
Load Program Status Word
Load Program Status Word Register
Exchange Program Status Register
Simulate Interrupt
Simulate Channel Program

If a program running in the protect mode attempts to execute a Privileged instruction, the instruction is not executed. The Processor saves the current PSW in Registers 14 and 15 of the register set 0. The illegal instruction new PSW at location X'000030' becomes the current PSW. The Location Counter of the old PSW contains the address of the Privileged instructions.

Illegal Instruction Interrupt

The illegal instruction interrupt cannot be disabled. The interrupt occurs whenever the Processor fetches an instruction word containing an operation code that is not one of those permitted by the system. The Processor saves the current PSW in Registers 14 and 15 of the register set 0. The illegal instruction new PSW from memory location X'000030' becomes the current PSW.

When the Processor encounters an illegal instruction, it makes no attempt to execute it. The Location Counter of the old PSW contains the address of the illegal instruction.

Supervisor Call Interrupt

This interrupt occurs as the result of the execution of a Supervisor Call instruction. This instruction provides a means for user level programs to communicate with system programs. The supervisor call interrupt is always enabled. When the Processor executes a Supervisor Call instruction, it:

Saves the current PSW in Registers 14 and 15 of the register set 0.

Places the address of the supervisor call parameter block (the second operand) in Register 13 of the register set 0.

Loads the current PSW status with the value contained at memory location X'000098', supervisor call new PSW status.

Loads the current PSW Location Counter from one of the supervisor call new PSW Location Counter locations (depending on the first operand).

STATUS SWITCHING INSTRUCTION FORMATS

The Status Switching instructions use the Kegister to Register (RR), and the Register and Indexed Storage (RX) instruction formats. In some cases, Load Program Status Word and Load Program Status Word Register, and the R1 field of the instruction has no significance and must be ZERO.

STATUS SWITCHING INSTRUCTIONS

The Status Switching instructions use the Register to Register (RR), and the Register and Indexed Storage (RX) instruction formats. In three instructions, Load Program Status Word, Load Program Status Word Register and Simulate Interrupt, the R1 field of the instruction has no significance and must be ZERO.

The instructions described in this section are:

LPSW	Load Program Status Word
LPSWR	Load Program Status Word Register
EPSR	Exchange Program Status Register
SINT	Simulate Interrupt
SVC	Supervisor Call

INSTRUCTION

Load Program Status Word (LPSW)

Assembler Notation		Op-Code	Format
LPSW	D2 (X2)	C2	RX1, RX2
LPSW	A2 (FX2, SX2)	C2	RX3

Operation

The 64 bit second operand becomes the current Program Status Word.

Condition Code

Determined by the new PSW (bits 28:31)

Programming Note

The quantity to be loaded into the current Program Status Word must be located in memory on a double word boundary.

This instruction is a privileged operation.

The R1 field of this instruction must be zero.

This instruction may be used to change register sets. The new set becomes active for execution of the next instructions.

INSTRUCTION

Load Program Status Word Register (LPSWR)

Assembler Notation		Op-Code	Format
LPSWR	R2	18	RR

Operation

The contents of the register specified by R2 replace Bits-0:31 of the current Program Status Word. The contents of the register specified by R2+1 replace Bits-32:63 of the current Program Status Word.

Condition Code

Determined by the new PSW (Bits 28:31)

Programming Note

The R1 field of this instruction must be zero.

This instruction may be used to change register sets. The new set becomes active for execution of the next instructions.

This instruction is a privileged operation.

The R2 field of this instruction may not specify a register greater than 14.

INSTRUCTION

Exchange Program Status Register (EPSR)

Assembler Notation		Op-Code	Format
EPSR	R1, R2	95	RR

Operation

Bits 0:31 of the current Program Status Word replace the contents of the register specified by R1. The contents of the register specified by R2 replace Bits 0:31 of the current Program Status Word.

Condition Code

Determined by the new PSW (Bits 28:31)

Programming Note

If R1 = R2, Bits 0:31 of the current PSW are copied into the register specified by R1, but otherwise remain unchanged.

This instruction may be used to change register sets. The new set becomes active for execution of the next instructions.

This instruction is a privileged operation.

INSTRUCTION

Simulate Interrupt (SINT)

Assembler Notation		Op-Code	Format
SINT	I2(X2)	E2	RI1

Operation

The least significant 10 bits of the second operand are presented to the interrupt handler as a device number. The device number is used to index into the interrupt service pointer table, simulating an interrupt request from an external device. The result is either an immediate interrupt or an auto driver channel operation.

Condition Code

Determined by the new PSW in case of immediate interrupt or determined by the way the auto driver channel operation terminates.

Programming Note

The R1 field of this instruction must contain zero.

This instruction is a privileged operation.

In the execution of this instruction, the Processor loads Registers 0:3 or 0:4 of the register set 0 as for a real interrupt request.

During the execution of this instruction, the device is addressed and the status byte is returned in register 3 of the register set 0.

In the event of instruction time-out, the V flag is set in the PSW, and register 3 of the register set 0 contains Y'00000004'.

INSTRUCTION

Supervisor Call (SVC)

Assembler Notation		Op-Code	Format
SVC	N, D2 (X2)	E1	RX1, RX2
SVC	N, A2 (FX2, SX2)	E1	RX3

Operation

The address of the second operand replaces Bits 8:31 of Register 13 of the register set 0. Bits 0:7 of this register are forced to ZERO. The current Program Status Word replaces the contents of Registers 14 and 15 of the register set 0. The fullword quantity located at X'000098' in memory replaces Bits 0:31 of the current Program Status Word. The four-bit N field is doubled and added to X'00009C'. The halfword quantity located at this address becomes the current Location Counter.

Condition Code

Determined by the new PSW (Bits 28:31)

Programming Note

The second operand must be located on a fullword boundary.

This instruction provides means of switching from the Protect Mode to the Supervisor Mode. It is used by the user program running under an Operating System to initiate certain functions in the Supervisor program. The second operand address, is normally a pointer to the memory location of the parameters the Supervisor program needs to complete the function specified. The type of Supervisor call is specified in the N field of the instruction. Sixteen different calls are provided for. Return from the Supervisor is made by executing an LPSWR instruction specifying the stored "Old" PSW in Registers 14, 15 of the register set 0 (LPSWR R14).

CHAPTER 7
INPUT/OUTPUT OPERATIONS

INTRODUCTION AND CONFIGURATION OF I/O SYSTEM

Input output (I/O) operations, as defined for the 32 bit series, provide a versatile means for the exchange of information between the Processor, memory, and external devices. Communication between the Processor and external devices is accomplished over the I/O Multiplexor Channel Bus (Byte or Halfword Modes). Data transfers over the Multiplexor Channel require Processor intervention, either programmed or automatic for each item transferred.

Direct data transfers between external devices and memory are accomplished over the Extended Direct Memory Access (EDMA) Bus, (Byte, Halfword or Burst Mode) and proceed independently of the Processor, so other program processing can proceed simultaneously. For more details refer to the following manuals:

1. EDMA Bus Universal Interface Instruction Manual, Publication Number 29-423
2. ESELCH Programming Manual, Publication Number 29-529

Burst mode data transfers over the EDMA Bus are possible only with the help of the EDMA Bus Universal Interface 02-361 which can handle data transfer rates up to six Megabytes per second between Local Memory and a custom designed I/O systems. In the burst mode, the originating device transmits the starting memory address and Burst Read or Burst Write command. This is followed by an arbitrary number of fullword data transmissions (up to six Megabytes/sec). Lower limit burst mode data transmission rate is 400 Kbytes/sec (10 microsec/fullword), below which bus control circuits assume the transmitter dead and abort the transfer.

DEVICE CONTROLLERS

The basic functions of all device controllers are:

1. To provide synchronization with the Processor and to provide device address recognition.
2. To transmit operational commands from the Processor to the device.
3. To translate device status into meaningful information for the Processor.
4. To request Processor attention when required.

In addition, controllers may generate parity, convert serial data to parallel, buffer incoming or outgoing data, or perform other device-dependent functions.

Device Addressing

The system design allows as many as 1,023 external devices. Each device must have its own unique device number or address. Device numbers may range from X'001' through X'3FF'. (Device number X'000' is not used.) The minimum system provides for 255 device numbers. Larger systems may have from 511 to 1,023 devices.

Processor/Controller Communication

Device controllers may be attached directly to the I/O Bus, or they may be attached to the I/O Bus indirectly through a Selector Channel. Communication between the Processor and controller is a bi-directional, request-response type of operation.

The Processor can initiate a communication, by sending the device address out onto the I/O Bus. When a controller recognizes the address, it returns a synchronization signal to the Processor, and remains ready to accept commands from the Processor. The Processor waits up to 35 microseconds for the synchronization signal. If no signal is received within this period, the Processor aborts the operation and notifies the controlling program. In this case, the status returned is X'04'. The condition code in the PSW, known as False Sync., is also set to X'4' (V flag = 1). Controller malfunction and software failure (incorrect device address) are the most common causes of this type of time-out.

A controller can initiate communication with the Processor by generating an attention signal. If the Processor is in the interruptable state as defined by Bit 17 of PSW, it temporarily suspends the normal "fetch instruction, execute, fetch next instruction" operation at the end of the execute phase, and transmits an acknowledge signal over the I/O Bus. The controller requesting attention responds with a synchronization signal, and transmits its device number to the Processor.

Device Priorities — External Interrupt ; Interrupt Queuing

Device requests for attention are asynchronous; therefore more than one request may be pending at any time. The system resolves these conflicts according to device priority, determined by the physical placement of the device controller on the I/O Bus. When two or more device controllers request attention at the same time, the controller "nearest" to the Processor in the RACK0/TACK0 priority wiring pattern captures the Acknowledge signal from the Processor and gets serviced first. All other interrupting controllers further down the line in priority must wait for the next Acknowledge signal from the Processor. Requests for attention remain queued until serviced by the Processor.

For details on standard and modified RACK0/TACK0 priority wiring patterns, see 01-078A20 (Installation section).

INTERRUPT SERVICE POINTER TABLE

Device requests for service may result in either an immediate interrupt or an Auto Driver Channel operation. The Processor chooses one of these options according to information contained in the Interrupt Service Pointer Table.

The Interrupt Service Pointer Table is an ordered list containing one entry for each possible device number in the system. The table starts at memory location X'0000D0' and contains a halfword entry for each device number in the system. For a minimum system, 255 device numbers, the table extends through memory location X'0002CF'; for a maximum system, the table extends through memory location X'0008CF' (1023 device numbers). The software controlling I/O operations must set up the table.

When, having acknowledged a request for service, the Processor receives the device address, it adds two times the device address to X'000D0'. The result is the address, within the table, of the entry reserved for the device requesting attention.

If the entry in the table is even (Bit 15 equals 0), the Processor takes an immediate interrupt and transfers control to the software routine at the address contained in the table. If the entry in the table is odd (Bit 15 equals 1), the Processor transfers control to the Auto Driver Channel, without interrupting the currently running program.

At the time the Processor transfers control to the software routine, the old PSW (current at the time of the device request) has been saved in Registers 0 and 1 of the register set 0. The device number is saved in Register 2 and the status in Register 3. The status portion of the current PSW has been forced to a value of X'00002800', thus switching to register set 0. Machine Malfunction Interrupt is enabled and all other interrupts disabled. The entry in the Interrupt Service Pointer Table has become the new Location Counter. (See Table 6-1.)

In using the device number presented by the controller to vector into the Interrupt Service Pointer Table, the Processor masks off the high order bits of the address as received from the I/O Bus. In a system with only 255 device numbers, the address is masked to eight bits. In a system with 1,023 device numbers the address is masked to 10 bits. The action preserves system integrity in the event that a hardware malfunction results in a device address greater than the maximum allowed in the system. (See Table 6-1.)

I/O INSTRUCTION FORMATS

The I/O instructions use the Register to Register (RR) and the Register and Indexed Storage (RX) instruction formats.

I/O INSTRUCTIONS

Following most I/O instructions, the V flag in the Condition Code indicates an instruction time-out. This means that the operation was not completed, either because the device did not respond at all, or because it responded incorrectly.

In the Sense Status and Block I/O instructions, the V flag can also mean examine status. To distinguish between these two conditions, the program should test Bits 0:3 of the device status byte. If all of these bits are ZERO, device time-out has occurred.

The instructions described in this section are:

SS	Sense Status	RBR	Read Block Register
SSR	Sense Status Register	WD	Write Data
OC	Output Command	WDR	Write Data Register
OCR	Output Command Register	WH	Write Halfword
RD	Read Data	WHR	Write Halfword Register
RDR	Read Data Register	WB	Write Block
RH	Read Halfword	WBR	Write Block Register
RHR	Read Halfword Register	AL	Autoload
RB	Read Block	SCP	Simulate Channel Program

INSTRUCTIONS

Output Command (OC)
Output Command Register (OCR)

Assembler Notation		Op-Code	Format
OC	R1, D2 (X2)	DE	RX1, RX2
OC	R1, A2 (FX2, SX2)	DE	RX3
OCR	R1, R2	9E	RR

Operation

Bits 22:31 of the register specified by R1 contain the 10 bit device address. The Processor addresses the device and transmits an eight-bit command byte from the second operand location to the device. Neither operand is changed.

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
0	1	0	0	Instruction time-out (FALSE SYNC) or EXAMINE status

Programming Note

In the RR format, Bits 24:31 of the register specified by R2 contain the device command.

These instructions are privileged operations.

INSTRUCTIONS

Sense Status (SS)
Sense Status Register (SSR)

Assembler Notation		Op-Code	Format
SS	R1, D2 (X2)	DD	RX1, RX2
SS	R1, A2 (FX2, SX2)	DD	RX3
SSR	R1, R2	9D	RR

Operation

Bits 22:31 of the register specified by R1 contain the 10 bit device address. The device is addressed and the eight bit device status is placed in the second operand location. The Condition Code is set equal to the right most four bits of the device status byte. The first operand is unchanged.

Condition Code

Bits 4:7 of the device status byte are copied into the Condition Code. See the appropriate device manual for a description of this status.

If the device is not in the system, condition code is set to 0100.

Programming Note

In the RR format, the device status byte replaces Bits 24:31 of the register specified by R2. Bits 0:23 are forced to zero.

These instructions are privileged operations.

INSTRUCTIONS

Read Data (RD)
Read Data Register (RDR)

Assembler Notation		Op-Code	Format
RD	R1, D2 (X2)	DB	RX1, RX2
RD	R1, A2 (FX2, SX2)	DB	RX3
RDR	R1, R2	9B	RR

Operation

Bits 22:31 of the register specified by R1 contain the 10 bit device address. The Processor addresses thc device. The device responds by transmitting an eight-bit data byte. This byte is placed in the second operand location.

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
0	1	0	0	Instruction time-out (FALSE SYNC) or EXAMINE status

Programming Note

In the RR format, the eight bit data byte replaces Bits 24:31 of the register specified by R2. Bits 0:23 of the register are forced to zero.

These instructions are privileged operations.

INSTRUCTIONS

Read Halfword (RH)
Read Halfword Register (RHR)

Assembler Notation		Op-Code	Format
RH	R1, D2 (X2)	D9	RX1, RX2
RH	R1, A2 (FX2, SX2)	D9	RX3
RHR	R1, R2	99	RR

Operation

Bits 22:31 of the register specified by R1 contain the 10 bit device address. The Processor addresses the device. If the device is halfword oriented, the Processor transmits 16 bits of data from the device to the second operand location. If the device is byte oriented, the Processor transmits two eight-bit bytes in successive operations.

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
0	1	0	0	Instruction time-out (FALSE SYNC) or EXAMINE status

Programming Note

In the RR format, the data received from a halfword device replaces Bits 16:31 of the register specified by R2. Bits 0:15 are forced to zero. The first byte of data from a byte device replaces Bits 16:23 of the register specified by R2. The second byte replaces Bits 24:31. Bits 0:15 are forced to ZERO.

If the device is byte-oriented, it must be capable of supplying both bytes without intervening status checks. Unlike the RB and RBR instructions, this instruction does not perform status checking between the two byte transfers.

In the RX format, the second operand must be located on a halfword boundary.

These instructions are privileged operations.

Read Block (RB)

Assembler Notation		Op-Code	Format
RB	R1, D2 (X2)	D7	RX1, RX2
RB	R1, A2 (FX2, SX2)	D7	RX3

Operation

Bits 22:31 of the register specified by R1 contain the 10 bit device address. Bits 8:31 of the fullword located at the second operand address contain the starting address of the data buffer. Bits 8:31 of the fullword located at the second operand address plus four contain the ending address of the data buffer.

The Processor transmits eight bit data bytes from the device to consecutive locations in the specified buffer.

Condition Code

Bits 4:7 of the device status byte are copied into the Condition Code. See the appropriate device manual for a description of this status.

If the device is not in the system, condition code is set to 0100.

Programming Note

The starting address must be less than, or equal to, the ending address. If the starting address is greater than the ending address, no transfer takes place and the Processor forces the Condition Code to ZERO. If the addresses are equal, one byte of data is transmitted.

The Processor is in a non-interruptable state during the transfer.

This instruction is a privileged operation.

The second operand must be located on a fullword boundary.

Read Block Register (RBR)

Assembler Notation		Op-Code	Format
RBR	R1, R2	97	RR

Operation

Bits 22:31 of the register specified by R1 contain the 10 bit device address. The register specified by R2 contains the starting address of the data buffer. The register specified by R2+1 contains the ending address of the data buffer.

The Processor transmits eight bit data bytes from the device to consecutive locations in the specified buffer.

Condition Code

Bits 4:7 of the device status byte are copied into the Condition Code. See the appropriate device manual for a description of this status.

If the device is not in the system, condition code is set to 0100.

Programming Note

The starting address must be less than, or equal to, the ending address. If the starting address is greater than the ending address, no transfer takes place and the Processor forces the Condition Code to ZERO. If the addresses are equal, one byte of data is transmitted.

The Processor is in a non-interruptable state during the transfer.

This instruction is a privileged operation.

INSTRUCTION

Write Data (WD)
Write Data Register (WDR)

Assembler Notation		Op-Code	Format
WD	R1, S2(X2)	DA	RX1, RX2
WD	R1, A2(FX2, SX2)	DA	RX3
WDR	R1, R2	9A	RR

Operation

Bits 22:31 of the register specified by R1 contain the 10 bit device address. The Processor addresses the device and transmits an eight data byte from the second operand location to the device. Neither operand is changed.

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
0	1	0	0	Instruction time-out (FALSE SYNC) or EXAMINE status

Programming Note

In the RR format, the eight bit data byte is contained in Bits 24:31 of the register specified by R2.

These instructions are privileged operations.

INSTRUCTION

Write Halfword (WH)
Write Halfword Register (WHR)

Assembler Notation		Op-Code	Format
WH	R1, D2 (X2)	D8	RX1, RX2
WH	R1, A2 (FX2, SX2)	D8	RX3
WHR	R1, R2	98	RR

Operation

Bits 22:31 of the register specified by R1 contain the 10 bit device address. The Processor addresses the device. If the device is halfword oriented, the Processor transmits 16 bits of data from the second operand location to the device. If the device is byte oriented, the Processor transmits two eight bit data bytes in successive operations.

Condition Code

C	V	G	L	
0	0	0	0	Operation successful
0	1	0	0	Instruction time-out (FALSE SYNC) or EXAMINE status

Programming Note

In the RR format, the data transmitted to a halfword device comes from Bits 16:31 of the register specified by R2. The first byte of data transmitted to a byte device comes from Bits 16:23 of the register specified by R2, the second byte, from Bits 24:31.

If the device is byte-oriented, it must be capable of accepting both bytes without intervening status checks. Unlike the WB and WBR instructions, this instruction does not perform status checking between the two byte transfers.

In the RX format, the second operand must be located on a halfword boundary.

These instructions are privileged operations.

INSTRUCTION

Write Block (WB)

Assembler Notation		Op-Code	Format
WB	R1, D2 (X2)	D6	RX1, RX2
WB	R1, A2 (FX2, SX2)	D6	RX3

Operation

Bits 22:31 of the register specified by R1 contain the 10 bit device address. Bits 8:31 of the fullword located at the second operand address contain the starting address of the data buffer. Bits 8:31 of the fullword located at the second operand address plus four contain the ending address or the data buffer.

The Processor transmits eight bit data bytes from consecutive locations in the specified buffer to the device.

Condition Code

Bits 4:7 of the device status byte are copied into the Condition Code. See the appropriate device manual for a description of this status.

If the device is not in the systen , the condition code is set to 0100.

Programming Note

The starting address must be less than, or equal to, the ending address. If the starting address is greater than the ending address, no transfer takes place and the Processor forces the Condition Code to ZERO. If the addresses are equal, one byte of data is transmitted.

The Processor is in a non-interruptable state during the transfer.

This instruction is a privileged operation.

The second operand must be located on a fullword boundary.

INSTRUCTION

Write Block Register (WBR)

Assembler Notation		Op-Code	Format
WBR	R1, R2	96	RR

Operation

Bits 22:31 of the register specified by R1 contain the 10 bit device address. The register specified by R2 contains the starting address of the data buffer. The register specified by R2+1 contains the ending address of the data buffer.

The Processor transmits eight bit data bytes from consecutive locations in the specified buffer to the device.

Condition Code

Bits 4:7 of the device status byte are copied into the Condition Code. See the appropriate device manual for a description of this status.

If the device is not in the system, the condition code is set to 0100.

Programming Note

The starting address must be less than, or equal to, the ending address. If the starting address is greater than the ending address, no transfer takes place and the Processor forces the Condition Code to ZERO. If the addresses are equal, one byte of data is transmitted.

The Processor is in a non-interruptable state during the transfer.

This instruction is a privileged operation.

INSTRUCTION

Autoload (AL)

Assembler Notation		Op-Code	Format
AL	D2 (X2)	D5	RX1, RX2
AL	A2 (FX2, SX2)	D5	RX3

Operation

The Autoload instruction loads memory with a block of data from a byte oriented input device. The data is read a byte at a time and stored in successive memory locations starting with location X'000080'. If the status is bad, the operation is terminated with V, G or L flags set. The last byte is loaded into the memory location specified by the address of the second operand. Any blank or zero bytes that are input prior to the first non-zero byte are considered to be leader and are ignored. All other zero bytes are stored as data. The eight bit input device address is specified by memory location X'000078'. The device command code is specified by memory location X'000079'.

Condition Code

C	V	G	L	
0	0	0	0	Operation successful or aborted.
X	1	X	X	Examine status or time-out
X	X	1	X	End of medium
X	X	X	1	Device unavailable

Programming Note

This instruction may only be used with devices whose addresses are less than, or equal to, X'FF'.

The R1 field of this instruction must be ZERO.

This instruction is a privileged operation.

The starting and ending addresses for this instruction are relocatable. Users should disable the Memory Access Controller before attempting to use this instruction.

If the second operand is less than X'80' the operation is aborted.

Example:

Assembler Notation			Machine Code		Comments
LOAD	AL	X'CF'	D500	00CF	Autoload program from X'80' to X'CF'
	B	X'80'	4300	0080	Jump to the program loaded
	.				
	.				
	.				
	.				
BINDV	DC	X'0294'	0294		Load using Teletypewriter tape reader This is location X'00078'

INSTRUCTION

Simulate Channel Program (SCP)

Assembler Notation		Op-Code	Format
SCP	R1, D2 (X2)	E3	RX1, RX2
SCP	R1, A2 (FX2, SX2)	E3	RX3

Operation

The second operand address is the address of a Channel Command Block (CCB). The buffer switch bit of the Channel Command Word (CCW) specifies the buffer to be used for the data transfer. If this bit is set, Buffer 1 is used. If it is reset, Buffer 0 is used. If the byte count field of the current buffer is positive, the V flag in the Condition Code is set, and the next sequential instruction is executed. If the byte count field is not positive, the following data transfer operation is performed.

If the Channel Command Word specifies read, a byte of data is moved from Bits 24:31 of the register specified by R1 to the appropriate buffer location. If the Channel Command Word specifies write, a byte of data is moved from the appropriate buffer location to Bits 24:31 of the register specified by R1. Bits 0:23 are forced to ZERO.

After a byte has been transferred, the count field of the appropriate buffer is incremented by one. If the count field is now positive, and if the last bit of the CCW is reset, the buffer switch bit of the CCW is complemented.

Condition Code

C	V	G	L	
0	0	0	0	Count field is now ZERO
0	0	0	1	Count field is now less than ZERO
0	0	1	0	Count field is now greater than ZERO
0	1	0	0	Count field was positive

Programming Note

The second operand must be located on a fullword boundary.

This instruction is a privileged operation.

CONTROL OF I/O OPERATIONS

The design of the 32 bit series I/O structure allows data transfers in any of several ways. The choice of which I/O method to use depends on the particular application and on the characteristics of the external devices. The primary methods of data transfer between the Processor and external devices are:

One byte or one halfword to or from any of the general registers.

One byte or one halfword to or from memory.

A block of data to or from memory under direct Processor control.

A block of data to or from memory under control of a Selector Channel or EDMA Universal Interface.

Multiplexed blocks of data to or from memory under control of the auto driver channel.

INTERDATA standard device controllers expect a predetermined sequence of commands to effect data transfers. These commands address the device, put it in the correct mode, and cause data to be transferred. Because all I/O instructions are privileged operations, I/O control programs must run in the Supervisor mode, Bit 23 of the current PSW reset. I/O control programs should disable immediate interrupts, controlled by PSW Bit 17.

STATUS MONITORING I/O

The simplest form of I/O programming is status monitoring I/O. In this mode of operation, only one device is handled at a time, and the Processor cannot overlap other operations with the data transfer. The sequence of operations in this type of programming is:

1. Address the device and set the proper mode (Output Command instruction).
2. Test the device status (Sense Status instruction).
3. Loop back to the Sense Status instruction until the status byte indicates that the device is ready (Conditional Branch instruction).
4. When the device is ready, transfer the data (Read or Write instruction).
5. If the transfer is not complete, branch back to the Sense Status instruction. If it is complete, terminate.

A variation on this type of programming makes use of the block I/O instructions. In this kind of programming, the program prepares the device and waits for it to become ready. It then executes a block I/O instruction. The Processor takes over control and completes the transfer, one byte at a time to or from memory. The Processor monitors device status during the transfer. Block transfers may be used only with byte oriented devices whose ready status is zero.

INTERRUPT DRIVEN I/O

Interrupt driven I/O allows the Processor to take advantage of the disparity in speed between itself and the external devices being controlled. With status monitoring, the Processor spends much of its time waiting for the device. With interrupt driven programming, the Processor can use much of this time to perform other functions. This kind of programming establishes at least two levels of operation. On one level are the interrupt service programs. On the other levels are the interruptable programs that run with the immediate interrupt enabled.

Before starting interrupt driven operations, the Interrupt Service Pointer Table must be set up. This table starts at memory location X'0000D0'. It must contain a halfword address entry for every possible device. The table is ordered according to device addresses in such a way that X'0000D0' plus two times the device address equals the memory address of the table entry reserved for that device. The value placed in the location reserved for a device is the address of the interrupt service routine for the device.

For example: if a console Teletype is connected at an address of X'02' and the interrupt routine resides in memory at address X'3000', the set up involves: writing X'3000' at memory location X'D4'. Note that X'D4' = X'D0' + 2 times the Teletype address.

Although there may be gaps in device address assignments, the interrupt service pointer table should be completely filled. Entries for non-existent devices can point to an error recovery routine. (This pi ecaution prevents system failure in the event of spurious interrupts caused by hardware malfunction or by improper use of the Simulate Interrupt instruction.)

The next step is to prepare the device for the transfer. This is done best with the immediate interrupt disabled. Once the table pointer has been set up, and the device prepared, the Processor can move on to an interruptable program.

When the device signals that it requires service, the Processor saves the current state, and transfers control to the location specified in the interrupt service pointer table. At this time, the current PSW has a status that indicates running state, machine malfunction interrupt enabled, I/O interrupts enabled and all other interrupts disabled. Registers 0 and 1 of the register set 0 contain the old PSW, indicating the status and location of the interrupted program. Register 2 of that set contains the device address. Register 3 contains the device status. The sequence of operation in this type of program is:

1. Set up the Interrupt Service Pointer Table to vector to error addresses for undefined plus devices.

2. Set up address of software interrupt handler routine at 2 times the device number plus X'D0' (X'D0' is starting address of Service Pointer table).

3. Set up software interrupt handler routine.

4. Set up the device and enable device interrupts.

5. Enable interrupts in PSW

The interrupt handler routine should:

1. Check the device status in Register 3, and if satisfactory,

2. Make the transfer, and

3. Return to the interrupted program by reloading the old PSW from Registers 0 and 1. (LPSWR R0)

The interrupt service routine should not enable the immediate interrupt. To do so allows other interrupt requests to be acknowledged, and the contents of Registers 0:4 would be lost. If it is necessary to enable the immediate interrupt, the routine should save the register set, and then enable the immediate interrupt.

SELECTOR CHANNEL I/O

The Selector Channel controls the transfer of data directly between high speed devices and memory. As many as 16 devices may be attached to the Selector Channel, only one of which may be operating at any one time. The advantage gained in using the Selector Channel is that other program processing may proceed simultaneously with the transfer of data between the external device and memory. This is possible because the Selector Channel accesses memory on a cycle stealing basis, which permits the Processor and the channel to share memory. In some cases, execution times of the program in progress may be affected, while in others, the effect is negligible. This depends upon the rate at which the Selector Channel and Processor compete for memory cycles.

The Selector Channel is linked to the Processor over the I/O Bus. It has its own unique device number which it recognizes when addressed by the Processor. Like other device controllers, it can request Processor attention through the immediate interrupt.

Selector Channel Devices

The Selector Channel has a private bus similar to the Processor's I/O Bus. Controllers for the devices associated with the Selector Channel are attached to this bus. When the Selector Channel is idle, its private bus is connected directly to the I/O Bus. If this condition exists, the Processor can address, command, and accept interrupt requests from the devices attached to the Selector Channel. When the Selector Channel is busy, this connection is broken. All communication between the Processor and devices on the Selector Channel are cut off. Any attempt by the Processor to address devices on the channel results in instruction time-out.

Selector Channel Operation

Two registers in the Selector Channel hold the current memory address and the final memory address. Before starting a Selector Channel operation, the control software, using Write instructions, places the address of the first byte of the data buffer in the current register and the address of the last byte in the final address register. During the data transfer, the channel increments the current address register by one for each byte transferred. When the current address equals the final address, the last byte has been transferred, and the channel terminates.

The Selector Channel accesses memory a halfword at a time. Therefore, the transfer must always involve an integer number of halfwords. The starting address of the data buffer must always be on an even byte (halfword) boundary. The ending address must always be on an odd byte boundary. The starting address must be less than the ending address.

Upon termination, the software can read back from the Selector Channel the address command in the current address register. If this address is less than the final address specified for the transfer, and if the buffer limits were properly checked before the transfer, then this condition indicates a device malfunction or an unusual condition within the device, for example, crossing a cylinder boundary on a disc.

Selector Channel Programming

The usual method of programming with the Selector Channel uses the immediate interrupt. The first step in the operation is to check the status of the Selector Channel. If it is not busy, the address of the termination interrupt service routine is placed in the location within the interrupt service pointer table reserved for the Selector Channel. Next the program should proceed as follows:

1. Give the Selector Channel a command to stop. This command initializes the Selector Channel's registers and assures the idle condition with the private bus connected to the I/O Bus.
2. Prepare the device for the transfer with whatever commands and information may be required.
3. Give the Selector Channel the starting and final addresses.
4. Give the Selector Channel the command to start.

With the Start command, the Selector Channel breaks the connection between its private bus and the Processor's I/O Bus, and provides a direct path to memory from the last device addressed over its bus. When the device becomes ready, the channel starts the transfer which proceeds to completion without further Processor intervention. Once the Start command has been given, the Processor can be directed to the execution of concurrent programs.

On termination, the channel signals the Processor that it requires service. The Processor subsequently takes an immediate interrupt, transferring control to the Selector Channel interrupt service routine. At this time, Registers 0:3 of the register set 0 are set up as for any other immediate interrupt.

AUTO DRIVER CHANNEL

The Auto Driver Channel provides a means for multiplexing block data transfers between memory and low or medium speed I/O devices. The operation of the channel is similar in some respects to interrupt driven I/O. The channel is activated upon a service request from a device on the I/O Bus. Upon receipt of a device request, the Processor uses the device number to index into the Interrupt Service Pointer Table. If the value contained in the table is even, the Processor transfers control to the interrupt service routine. If the value is odd, it transfers control to the Auto Driver Channel.

To the Auto Driver Channel, the address in the Interrupt Service Pointer Table is the address plus one (making it odd) of a Channel Command Block (CCB). The Channel Command Block is basically a channel program consisting of a description of the operation to be performed, and a list of parameters associated with the operation. In addition to the functions of Read and Write, the channel can (a) translate characters, (b) test device status, (c) chain buffers, (d) calculate longitudinal and cyclic redundancy check values, and (e) transfer control to software routines to take care of unusual situations.

CHANNEL COMMAND BLOCK

The Channel Command Block (CCB), as shown in Figure 7-1, consists of a Channel Command Word (16 bits) that describes the function, count fields (16 bits each) for two buffers, final addresses (32 bits each) for two buffers, a check word (16 bits) for the longitudinal or cyclic redundancy check, the address (32 bits) of a translation table, and the address (16 bits) of a software routine.

	0	15
0	CHANNEL COMMAND WORD	(HALFWORD)
2	BUFFER 0 BYTE COUNT	(HALFWORD)
4	BUFFER 0 END ADDRESS	(FULLWORD)
8	CHECK WORD	(HALFWORD)
10	BUFFER 1 BYTE COUNT	(HALFWORD)
12	BUFFER 1 END ADDRESS	(FULLWORD)
16	TRANSLATION TABLE ADDRESS	(FULLWORD)
20	SUBROUTINE ADDRESS	(HALFWORD)

Figure 7-1. Channel Command Block

Just as there may be many interrupt service routines ready at any time to service device requests, there may be many channel command blocks in the system ready to handle data transfers as required. Each channel command block must start on a fullword boundary. The address plus one of the channel command block must be placed in the interrupt service pointer table location for the device involved in the transfer.

Subroutine Address

When the channel transfers control to the software subroutine whose address is contained in the Channel Command Block, Registers 0:4 of the register set 0 have already been set up by the Processor to contain the old PSW, the device number, the device status, and the address of the Channel Command Block. The current PSW status specifies run state, machine malfunction interrupt enabled, immediate interrupt enabled, and all other interrupts disabled.

The channel transfers control to the subroutine either unconditionally (controlled by a bit in the Channel Command Word), or because of bad device status, or because it has reached the limit of a buffer. It indicates its reason for transferring control by adjusting the Condition Code as follows.

C	V	G	L	
0	0	0	0	Unconditional transfer
0	0	0	1	Bad status
0	0	1	0	Buffer limit

The subroutine address in the CCB is a 16 bit address. Because of this, the subroutine, or at least the first instruction of the subroutine, must reside in the first 64KB of memory.

Buffers

There is space in the CCB to describe two data buffer areas. The data areas may be located anywhere in memory. The limits of each data area are described by an address field and a count field. The address field contains the address of the last byte in the data area. This is a 24 bit address, right justified in the fullword provided. If the device being controlled is a halfword device, the final address must be odd. If the device is a byte device, the address may be either odd or even.

The count field, in most operations, contains a negative number whose absolute value is equal to one less than the number of bytes to be transferred. The one exception is the case of a single byte transfer, where the count field contains ZERO.

During data transfers, the channel adds the value contained in the count field to the final address to obtain the current address. It makes the transfer, referencing the current address, then increments the value in the count field by one for a byte device or by two for a halfword device. When the count field becomes positive, i.e., greater than zero, the channel sets the G flag in the Condition Code and transfers control to the specified software subroutine. If the count field is positive upon channel activation, the Channel makes no transfer and returns control to the processor with Condition Code = 0010 (G=1).

Translation

The translation feature is available only for byte devices. If this operation is specified, the fullword provided in the Channel Command Block must contain the 24 bit address, right justified, of a translation table. The table, which must start on a halfword boundary, can contain up to 256 halfword entries. During data transfers, the channel multiplies the data byte by two and adds this value to the translation table address. The result is the address within the translation table of the halfword corresponding to the data byte.

The channel references this location, and, if Bit 0 of the halfword is a one, it substitutes Bits 8:15 of the halfword for the data byte and proceeds with the operation. If Bit 0 of the halfword is a ZERO, the channel:

> Does not increment the byte count for the appropriate buffer.
>
> Puts the data byte, untranslated, in Bits 24:31 of Register 3, of the register set 0.
>
> Forces Bits 0:23 of Register 3 to ZERO.
>
> Multiplies the value contained in the translation table by two, and transfers control to the software routine located at this address.

Upon transfer to the translation subroutine, Registers 0 and 1 contain the old PSW. Register 2 contains the device number. Register 3 contains the untranslated character. Register 4 contains the address of the Channel Command Block. The current PSW indicates run state, machine malfunction interrupt enabled, immediate interrupt enabled and all other interrupts disabled. The Condition Code is zero.

Check Word

If either longitudinal or cyclic redundancy checking is required, the check word in the Channel Command Block contains the accumulated value. The initial value for the check word is usually zero. (There are data dependent exceptions, e.g., where initial characters are not to be included in the check.) The longitudinal check is an Exclusive OR of the character with the check word. The cyclic check uses the formula for CRC 16:

$$X^{16} + X^{15} + X^{2} + 1$$

If the Data Communication Option is equipped, the cyclic check may optionally use the formula for CRC SDLC:

$$X^{16} + X^{12} + X^{5} + 1$$

On input, if both redundancy checking and translation are required, the character is translated first, then the cyclic redundancy check is done using the original character input rather than the translated character. On output, the character is translated first. Redundancy checking may be used only with byte devices.

Channel Command Word

The Channel Command Word (CCW), as shown in Figure 7-2, consists of two parts. Bits 0:7 contain a status mask. Bits 8:15 describe the channel operation.

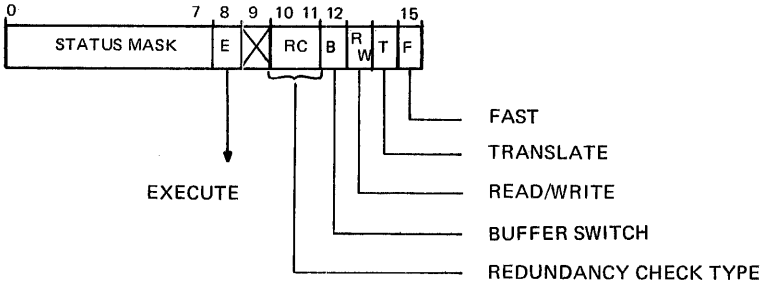


Figure 7-2. Channel Command Word

Status Mask

On every channel operation involving a data transfer, the status mask is ANDed with the device status. This operation does not change the status mask. If the result is zero, the channel proceeds with the operation. If the result is non-zero, the channel sets the L flag in the Condition Code, and transfers control to the specified software subroutine.

Execute Bit (E)

If this bit is reset, the channel unconditionally transfers control to the specified subroutine, without taking any other action. The Condition Code is zero. If this bit is set, the channel continues with the operation as specified in the Channel Command Word.

Fast Bit (F)

If this bit is set, the channel performs the I/O transfer in the fast mode. In the fast mode, buffer chaining, redundancy checking, and translation are not allowed. This bit must be set for halfword devices. If this bit is set, Buffer 0 is always used.

Read/Write Bit (R/W)

This bit indicates the type of operation. If this bit is reset, a byte or a halfword is inp ut from the device. If this bit is set, a byte or a halfword is output to the device.

Translate Bit (T)

If this bit is set, and the fast bit reset, the channel translates the data byte.

Redundancy Check Type Bits (RC)

These two encoded bits specify the type of redundancy check required. The following is a list of valid redundancy check specifications. These bits are ignored if the part bit (bit 15) is set. CRC SDLC can be specified only if the Data Communication option is installed.

BIT 10	BIT 11	Redundancy Check Type
0	0	LRC
0	1	BISYNC CRC
1	0	Reserved - Should Not be Specified
1	1	SDLC CRC - Should Only be Specified if Data Communication Option Present

Buffer Switch Bit (B)

When the fast bit is reset, this bit specifies which of the two buffers is to be used for the transfer. If this bit is reset, Buffer 0 is used. If it is set, Buffer 1 is used. The channel chains buffers when the count field becomes positive. It does this by complementing the buffer switch bit before transferring control to the specified software routine.

Valid Channel Command Codes

The following is a list of valid codes for the Channel Command Word. Note that only the first three may be used with halfword devices.

Channel Command Word 8:15

Hexadecimal	Binary	Meaning
00	00000000	Transfer to subroutine
81	10000001	Read fast mode
85	10000101	Write, fast mode
80	10000000	LRC, Buffer 0, Read
82	10000010	LRC, Buffer 0, Read, translate
84	10000100	LRC, Buffer 0, Write
86	10000110	LRC, Buffer 0, Write, translate
88	10001000	LRC, Buffer 1, Read
8A	10001010	LRC, Buffer 1, Read, translate
8C	10001100	LRC, Buffer 1, Write
8E	10001110	LRC, Buffer 1, Write, translate
90	10010000	CRC BISYNC, Buffer 0, Read
92	10010010	CRC BISYNC, Buffer 0, Read, translate
94	10010100	CRC BISYNC, Buffer 0, Write
96	10010110	CRC BISYNC, Buffer 0, Write, translate
98	10011000	CRC BISYNC, Buffer 1, Read
9A	10011010	CRC BISYNC, Buffer 1, Read, translate
9C	10011100	CRC BISYNC, Buffer 1, Write
9E	10011110	CRC BISYNC, Buffer 1, Write, translate
B0	10110000	CRC SDLC, Buffer 0, Read
B2	10110010	CRC SDLC, Buffer 0, Read, translate
B4	10110100	CRC SDLC, Buffer 0, Write
B6	10110110	CRC SDLC, Buffer 0, Write, translate
B8	10111000	CRC SDLC, Buffer 1, Read
BA	10111010	CRC SDLC, Buffer 1, Read, translate
BC	10111100	CRC SDLC, Buffer 1, Write
BE	10111110	CRC SDLC, Buffer 1, Write, translate

General Auto Driver Channel Programming Procedure (See Figure 7-3.)

1. Set up Interrupt Service Pointer Table to vector to error routines for undefined devices.

2. Set up address of Channel Command Word + 1 (odd) in table at 2 times Device number plus X'D0' (start of Interrupt Service Pointer Table)

3. Set up complete Channel Command Block.

4. Set up device and enable device interrupt.

5. Enable interrupts in PSW (Auto Driver Channel finishes operation).

6. Check for good termination of Auto Driver Channel.

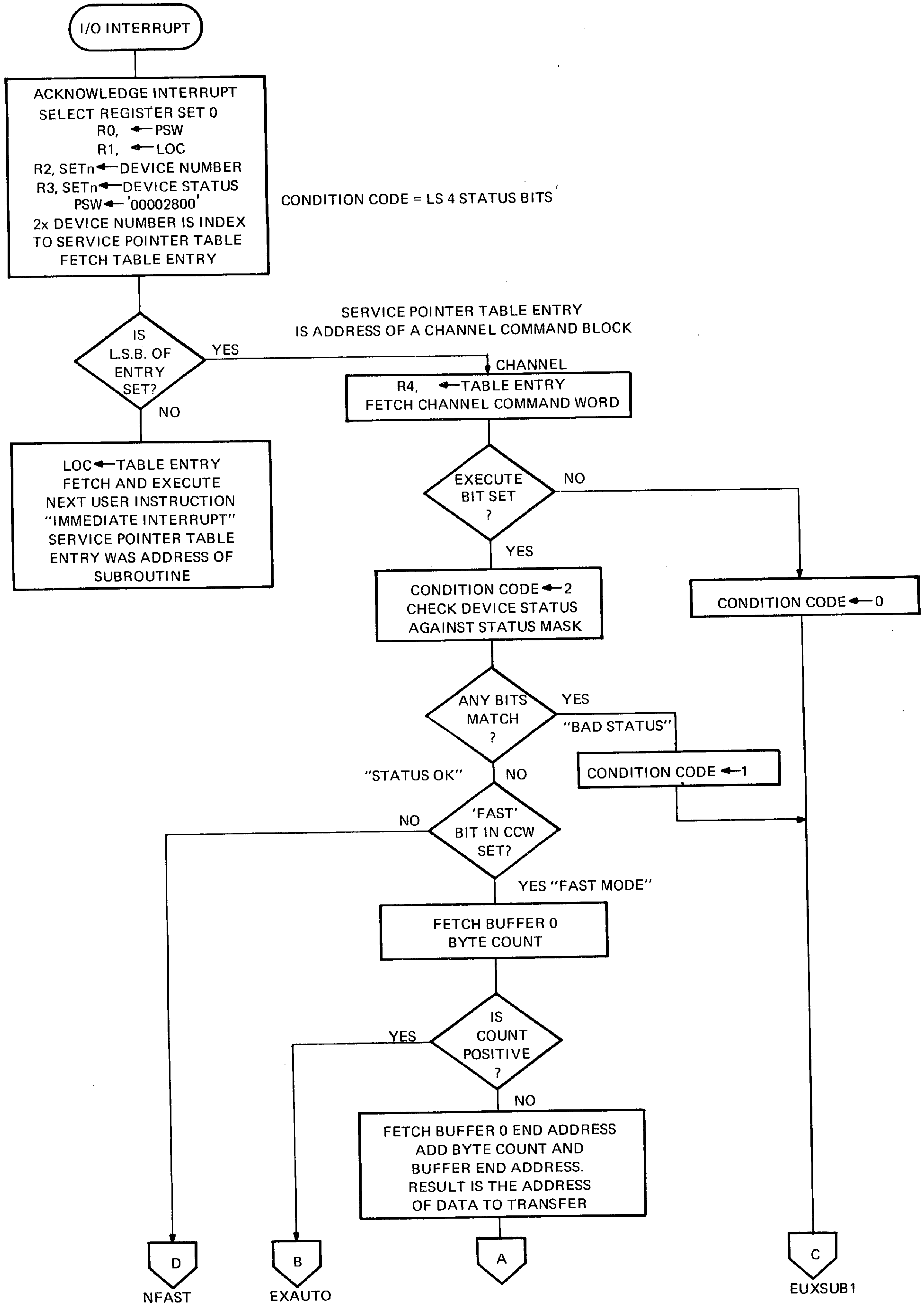


Figure 7-3. Microcode Flowchart of Auto Driver Channel (Sheet 1 of 3)

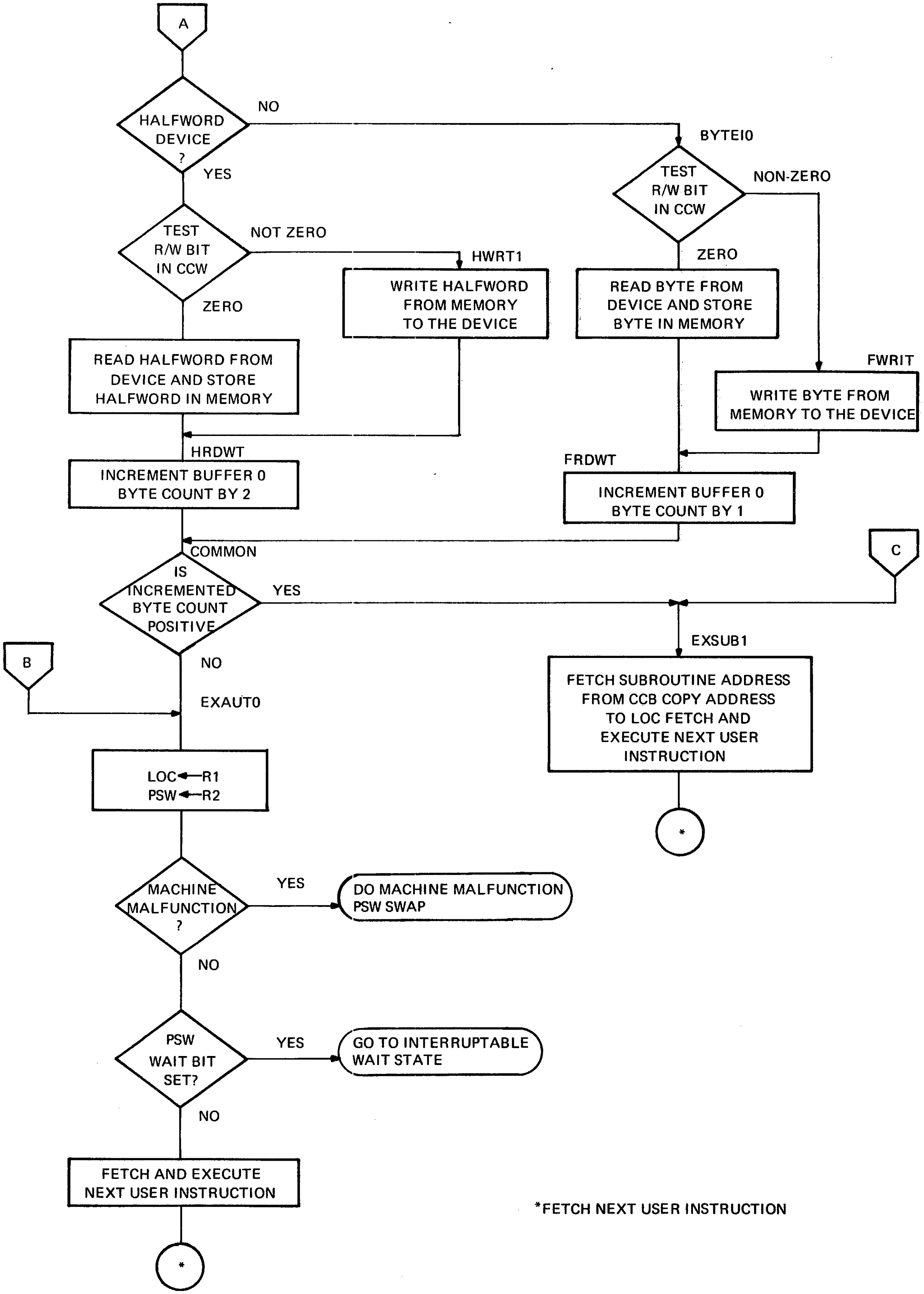

A
HALFWORD DEVICE ?
NO
YES
BYTEIO
TEST R/W BIT IN CCW
NON-ZERO
ZERO
TEST R/W BIT IN CCW
NOT ZERO
HWRT1
ZERO
WRITE HALFWORD FROM MEMORY TO THE DEVICE
READ BYTE FROM DEVICE AND STORE BYTE IN MEMORY
READ HALFWORD FROM DEVICE AND STORE HALFWORD IN MEMORY
FWRIT
WRITE BYTE FROM MEMORY TO THE DEVICE
HRDWT
INCREMENT BUFFER 0 BYTE COUNT BY 2
FRDWT
INCREMENT BUFFER 0 BYTE COUNT BY 1
COMMON
C
IS INCREMENTED BYTE COUNT POSITIVE
YES
EXSUB1
B
NO
EXAUTO
FETCH SUBROUTINE ADDRESS FROM CCB COPY ADDRESS TO LOC FETCH AND EXECUTE NEXT USER INSTRUCTION
LOC←R1
PSW←R2
*
MACHINE MALFUNCTION ?
YES
DO MACHINE MALFUNCTION PSW SWAP
NO
PSW WAIT BIT SET?
YES
GO TO INTERRUPTABLE WAIT STATE
NO
FETCH AND EXECUTE NEXT USER INSTRUCTION
*FETCH NEXT USER INSTRUCTION
*

Figure 7-3. Microcode Flowchart of Auto Driver Channel (Sheet 2 of 3)

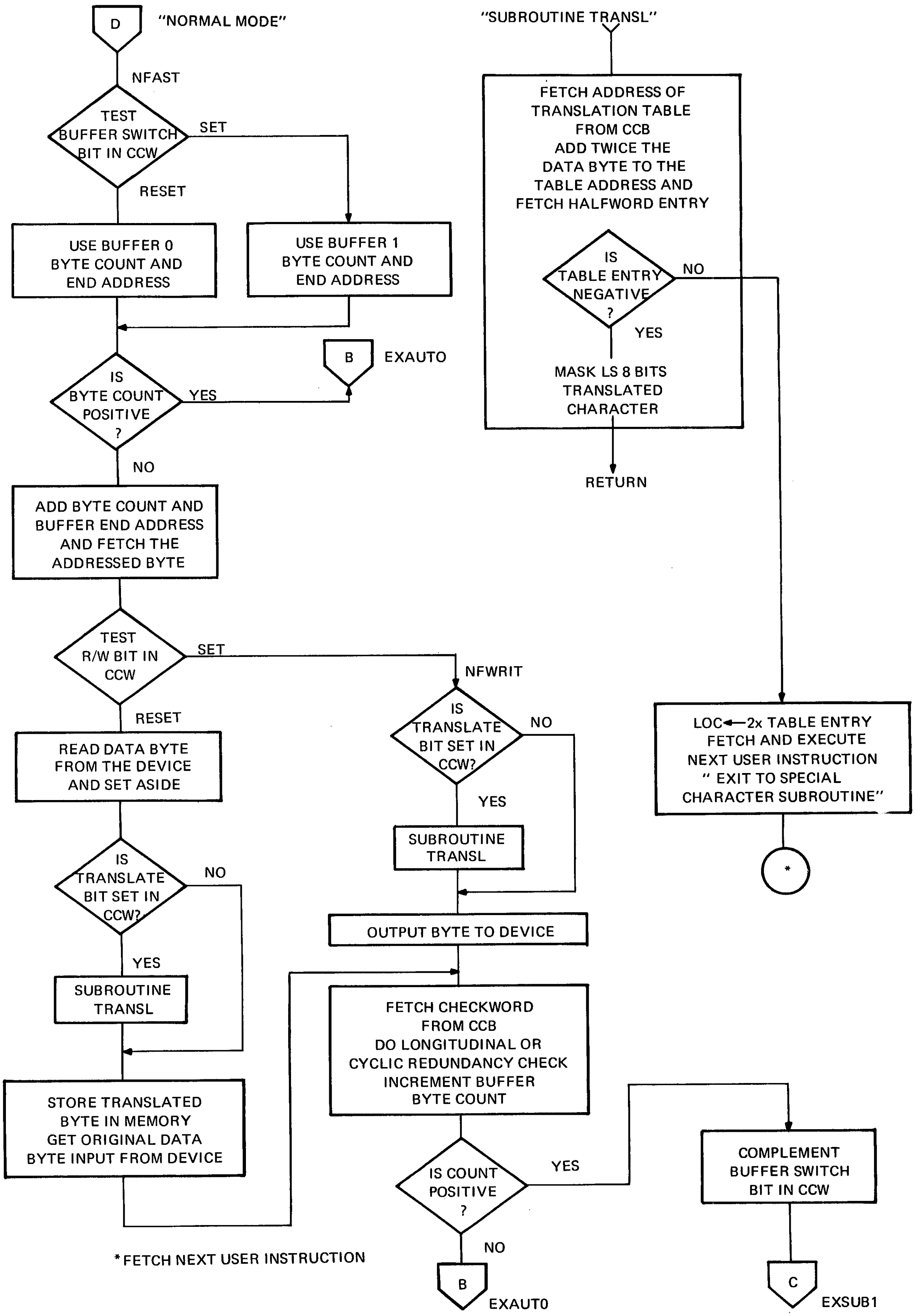

D
"NORMAL MODE"
NFAST
TEST BUFFER SWITCH BIT IN CCW
SET
RESET
USE BUFFER 0 BYTE COUNT AND END ADDRESS
USE BUFFER 1 BYTE COUNT AND END ADDRESS
B
EXAUTO
IS BYTE COUNT POSITIVE ?
YES
NO
ADD BYTE COUNT AND BUFFER END ADDRESS AND FETCH THE ADDRESSED BYTE
TEST R/W BIT IN CCW
SET
RESET
NFWRIT
READ DATA BYTE FROM THE DEVICE AND SET ASIDE
IS TRANSLATE BIT SET IN CCW?
NO
YES
SUBROUTINE TRANSL
STORE TRANSLATED BYTE IN MEMORY GET ORIGINAL DATA BYTE INPUT FROM DEVICE
IS TRANSLATE BIT SET IN CCW?
NO
YES
SUBROUTINE TRANSL
OUTPUT BYTE TO DEVICE
FETCH CHECKWORD FROM CCB DO LONGITUDINAL OR CYCLIC REDUNDANCY CHECK INCREMENT BUFFER BYTE COUNT
IS COUNT POSITIVE ?
YES
NO
B
EXAUTO
COMPLEMENT BUFFER SWITCH BIT IN CCW
C
EXSUB1
"SUBROUTINE TRANSL"
FETCH ADDRESS OF TRANSLATION TABLE FROM CCB ADD TWICE THE DATA BYTE TO THE TABLE ADDRESS AND FETCH HALFWORD ENTRY
IS TABLE ENTRY NEGATIVE ?
NO
YES
MASK LS 8 BITS TRANSLATED CHARACTER
RETURN
LOC←2x TABLE ENTRY FETCH AND EXECUTE NEXT USER INSTRUCTION " EXIT TO SPECIAL CHARACTER SUBROUTINE"
*
*FETCH NEXT USER INSTRUCTION

Figure 7-3. Microcode Flowchart of Auto Driver Channel (Sheet 3 of 3)

CHAPTER 8
MEMORY MANAGEMENT

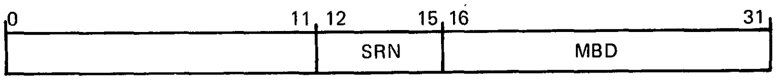
The 02-348 Memory Access Controller (MAC) is an optional auxiliary module available with Model 7/32. The MAC provides memory relocation and protection. The MAC is a device which monitors all memory accesses. Under program control, it can (a) translate the address of a memory access from a 20-bit program address to a 20-bit physical address, (b) prevent write access to a block of memory, (c) reject instruction execution from a block of memory or (d) detect an invalid memory access.

The throughput between the Processor and local memory or between the Selector Channel and local memory is not affected by the use of the MAC.

In an operating system environment, the operation of the MAC is completely transparent to most programs. It is very similar to a peripheral device in that only the operating system modules directly responsible for its operation need be aware of its existence.

PROGRAM ADDRESS SPACE

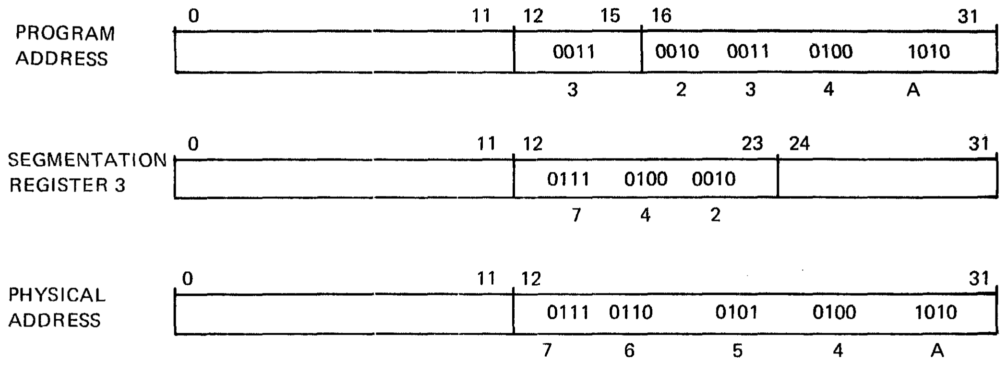
The MAC allows an Operating System to provide support to user programs in such a way that the program can be coded as if some subset of available memory, starting at address 0, were available to the program. The range of addresses thus referenced by the program is called the Program Address Space. At load time, the MAC can be used to map this program address space into the available physical memory addresses so that any program address, referenced during the program execution, is translated (relocated) to the correct physical address before memory is accessed. The MAC interprets the Program Address as follows



SRN: SEGMENTATION REGISTER NUMBER
MBD: MEMORY BLOCK DISPLACEMENT

RELOCATION

The relocation of program address to physical address is accomplished through the relocation/protection bit (bit 21) of the Program Status Word and the 16 Segmentation Registers of the MAC. If the relocation/protection bit of the PSW is reset, the MAC provides no translation of the addresses. If the relocation/protection bit of the PSW is set, the MAC assumes that all memory accesses are program addresses which must be relocated to physical addresses. Before the relocation/protection bit of the PSW is set, the MAC Segmentation Registers must be loaded with the appropriate mapping of the program to physical address (see below). The MAC Segmentation Register describes the starting address and length of a block of physical memory allocated to the program address space. These blocks must start on a 256 byte boundary and may be up to 64K bytes long.



Address calculation:		X'0234A'	Memory block displacement
	+	X'74200'	Memory block starting address
		X'7654A'	Physical memory address

When the relocation/protection bit of the PSW is set, the program address is relocated as follows:

Program address Bits 12:15 select one of the segmentation Registers. In the example above, segmentation Resister 3 is selected.

Segmentation Register Bits 12:23 specify starting address of the block of memory. In the illustration above, X'742' means that the memory block starting address is X'74200'.

Program address Bits 16:31 contain the memory block displacement.

The block displacement is added to the memory block starting address to obtain physical memory address.

PROTECTION

In addition to describing a block of physical addresses, each Segmentation Register can be used to limit the type of access to the described block of addresses. Five types of protection are provided by the MAC when the relocation/protection bit of the current PSW is set:

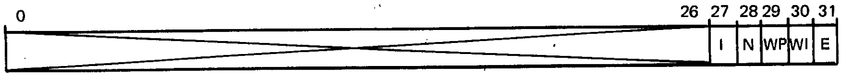
if the presence bit (Bit 27) is reset in the Segmentation Register selected by Bits 12:15 of the Program address (non-present address), or

if the write-protect bit (Bits 25:26 = 01 or 11) is set in the Segmentation Register selected by Bits 12:15 of the program address, and an attempt is made to store into the addressed memory (write protection violation), or

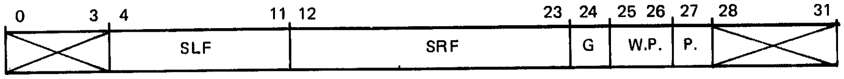
if write/interrupt protection bit (Bits 25:26 = 10) is set in the Segmentation Register selected by bits 12:15 of the program address and a store is made into the addressed memory (write/interrupt protection violation), or

if the execution-protection bit (Bit 24) is set in the Segmentation Register selected by Bits 12:15 of the program address and an instruction fetch is being attempted from the addressed memory (execute protection violation), or

if the value of Bits 24:31 of the program address is larger than the limit described in the Segmentation Register selected by Bits 12:15 of the program address (invalid address), then a Relocation/Protection Fault interrupt is generated. The MAC status register contains the reason for the interrupt (see below).



INTERRUPT STATUS REGISTER



SEGMENTATION REGISTER

In the case of an execution protection violation, write protection violation or invalid address, if the interrupt generated by the MAC cannot be accepted immediately by the Processor, the controller continues to operate, but all Write operations are changed to read operations until the interrupt is cleared. In the case of write/interrupt protect violation, the store operation is allowed to complete and then an interrupt is generated. A MAC interrupt condition is cleared by any reference to the MAC interrupt status register, however, only a store instruction will clear the status register.

EXAMPLE:

The effect of the MAC is best illustrated by an example of a program executing under operating system control.

Assume that the program consists of:

main program coded as if addresses 0 through 2FFF are available and a program entry address of 100. (Program Address Space = 12K)

a subroutine coded as if addresses F0000 through F1FFF are available. (Program Address Space = 8K)

a data area which is initialized by some other program and which is contained at addresses A0000 through AFFFF. This area is to be write and execute protected. (Program Address Space = 64K)

The operating system executes with the relocation/protection bit of the PSW reset so that no address relocation or protection is in effect.

Assume that the main program, subroutine and data area are loaded into physical memory starting at addresses 21000, F000, 13000 respectively. Before passing control to the example program, the operating system:

sets the start address of Segmentation Registers 0, 10 and 15 to 21000, 13000 and 0F000 respectively.

resets the presence bit in the remaining Segmentation Registers.

sets the limits of Segmentation Registers 0, 10 and 15 to 47, 255 and 31 blocks respectively.

sets write and executes protection in Segmentation Register 10.

0 3	4 11	12 23	24	25 26	27	28 31
	SLF	SRF	E	W.P.	P.	

SEGMENTATION REGISTER FIELDS

SEGMENTATION REGISTER 0:

0 3	4	11	12		23	24 27	28 31
	0010	1111	0010	0001	0000	0001	
0	2	F	2	1	0	1	0

SEGMENTATION REGISTER 10:

0 3	4	11	12		23	24 27	28 31
	1111	1111	0001	0011	0000	1011	
0	F	F	1	3	0	B	0

SEGMENTATION REGISTER 15:

0 3	4	11	12		23	24 27	28 31
	0001	1111	0000	1111	0000	0001	
0	1	F	0	F	0	1	0

SEGMENTATION REGISTERS 1,2,3,4,5,6,7,8,9,11,12,13 & 14:

0 3	4	11	12		23	24 27	28 31
	0000	0000	0000	0000	0000	0000	
0	0	0	0	0	0	0	0

The program can then be started by loading a PSW with relocation/protection bit of the status portion set and a location counter of 100. A relocation/protection fault interrupt occurs if:

an attempt is made to reference 30000. (Presence bit reset in selected Segmentation Register, i.e., Segmentation Register 3.)

an attempt is made to store into A0100. (Write protect set in selected Segmentation Register, i.e., Segmentation Register 10.)

an attempt is made to branch to A0000. (Execute protect set in selected Segmentation Register, i.e., Segmentation Register 10.)

an attempt is made to reference F3000. (Value of Bits 15:31 of program address (30000) is larger than the limit field of Segmentation Register 15 (32 256 byte blocks or 2000).

An attempt to reference 100, F1200 or A0001 results in an access to 21100, 10200 or 13001 respectively.

MAC REGISTERS

The MAC has 17 hardware registers referred to as Base Registers. There are 16 Segmentation Registers and 1 Interrupt Status Register. These registered are accessed through the assigned memory locations.

The 256 bytes starting at the first 256 byte boundary above the Interrupt Service Pointer Table, are dedicated to the MAC.

MAX NUMBER OF DEVICES	DEDICATED MAC LOCATIONS
256	300 – 3FF
512	500 – 5FF
1024	900 – 9FF

The MAC Registers are assigned to the dedicated locations as follows (for 256 maximum number of devices):

Segmentation Register				Memory Location
		0	-	300
"	"	1	-	304
"	"	2	-	308
"	"	3	-	30C
"	"	4	-	310
"	"	5	-	314
"	"	6	-	318
"	"	7	-	31C
"	"	8	-	320
"	"	9	-	324
"	"	10	-	328
"	"	11	-	32C
"	"	12	-	330
"	"	13	-	334
"	"	14	-	338
"	"	15	-	33C
Interrupt Status Register			-	340

Values are loaded into the MAC registers by storing the values into the appropriate assigned memory locations. Any attempt to read the dedicated MAC locations returns the value in the corresponding memory location except for the location assigned to the MAC Status Register. In general, manipulation of MAC registers is performed with the relocation/protection of the PSW reset. To summarize the manipulation of the MAC registers:

> The 68 bytes starting at the first 256 byte boundary above the Interrupt Service Pointer Table, are dedicated to the MAC (if present in the system).
>
> The value of a MAC register is changed by storing into the appropriate dedicated MAC location.
>
> The value of the MAC Status Register is read by loading from the appropriate dedicated MAC location.
>
> All attempts to read (load) from dedicated MAC locations return the value in the corresponding memory location , except for the MAC status register location.

MAC registers are manipulated, with the relocation/protection bit of the PSW reset , as follows:

The Segmentation Registers are set up by storing data into the appropriate assigned memory locations.

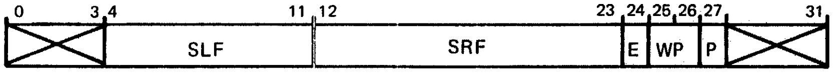
The Segmentation Registers cannot be read. Any attempt to read the dedicated MAC locations assigned for the Segmentation Registers returns the value in the corresponding memory locations. This value may be different than the actual (hardware) Segmentation Register value. To read the data which has been loaded into the Segmentation Registers, it is necessary to read the assigned locations after the registers have been loaded (with MAC disabled) and before the MAC is enabled. Under these conditions the assigned memory locations will contain the same data as the Segmentation Registers.

The Interrupt Status Register is cleared by writing any data into its assigned memory location.

The Interrupt Status Register can be read by reading its assigned memory location. This also clears the Interrupt Status Register.

Definition of MAC Register Fields

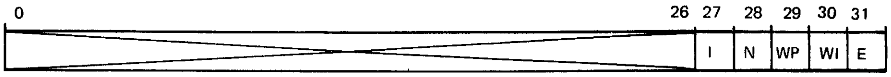
Segmentation Register



Each Segmentation Register is 32 bits wide.

Field	Bits	Meaning
	0-3	Unused - must be zero
SLF	4-11	Segment Limit Field, contains one less than the number of 256 byte blocks in the segment described by this register.
SRF	12-23	Segment Relocation Field - indicates the starting address of the segment described by this register (Starting address = SRF multiplied by X'100').
E	24	Execute protect bit - if set, instruction fetch from segment causes relocation/protection fault.
WP	25-26	Write protection field - encoded as follows: 00 -- no write protection 01 or 11 -- Write protected - attempt to store into segment causes relocation/protection fault - store is not executed. 10 -- Write/Interrupt protect - attempt to store into segment causes relocation/protect fault - store is executed.
P	27	Presence bit - if not set, selection of this register causes relocation/protection fault.
	28-31	Unused - must be zero.

INTERRUPT STATUS REGISTER



Field	Bits	Meaning
I	27	Invalid Address - value of bits 16:31 of program address greater than the limit specified by SLF in the selected Segmentation Register.
N	28	Non-present Address - present bit not set in selected segmentation register.
WP	29	Write Protect Violation - attempt to store into write protected segment.
WI	30	Write/interrupt protection violation - store into write/ interrupt protected segment.
E	31	Execute Protect Violation - instruction fetch attempt from execute protected segment.

The Interrupt Status Register is set by the MAC during generation of a relocation/protection fault interrupt. The first reference, load or store, to the memory location assigned to the interrupt status register following the interrupt, clears the interrupt condition from the MAC. The Relocation and protection interrupt handler should execute with the relocation/protection bit of the PSW reset and should clear the Interrupt Status Register by storing any fullword into the assigned memory location before exiting.

INITIALIZATION

Whenever the Initialize Switch (INT) on the display panel is depressed, or the processor is powered up, all segmentation, relocation, protection and MAC interrupts are disabled regardless of the state of bit 21 in the current PSW. The contents of the MAC segmentation registers must be restored by software after Power Fail.

The MAC remains disabled until a memory reference instruction is issued. At this time, the MAC is enabled or remains disabled depending on the condition of bit 21 of the current PSW.

CHAPTER 9
DATA COMMUNICATION INSTRUCTIONS

The Data Communications instructions are used to compute polynomial error check redundancy characters, as used by most data communications protocols. A high speed memory-to-memory move capability is also provided with this option. Communications protocols supported by this option include, but are not limited to, the following:

Binary Synchronous Communications (BISYNC or BSC) - IBM's widely accepted half-duplex protocol uses the CRC BISYNC error check polynomial ($x^{16} + x^{15} + x^2 + 1$).

Synchronous Data Link Control (SDLC) - IBM's new full-duplex protocol uses the CRC SDLC error check polynomial ($x^{16} + x^{12} + x^5 + 1$).

Advanced Data Communications Control Procedure (ADCCP) - ANSI's proposed National Standard full-duplex protocol uses CRC SDLC.

High Level Data Link Control (HDLC) - The ISO's International Standard full-duplex protocol uses CRC SDLC.

DATA COMMUNICATION INSTRUCTION FORMATS

The optional Data Communication instructions use the Register to Register (RR), and the Register and Indexed Storage (RX) formats.

DATA COMMUNICATION INSTRUCTIONS

PB	Process Byte
PBR	Process Byte Register
MPBSR	Move and Process Byte String Register

INSTRUCTION

Process Byte (PB)

Assembler Notation		Op-Code	Format
PB	R1, D2(X2)	62	RX1, RX2
PB	R1, A2(FX2, SX2)	62	RX3

Set Up

	0 7	8 15	16 23	24 31
R1	X	CHECK CODE	X	DATA BYTE

Bits 24:31 of the register specified by R1 contain the data byte to be processed. Bits 8:15 of the register specified by R1 contain a check code to indicate the type of processing. This byte is interpreted as follows:

X'00'	Cumulative check zero (CRC BISYNC)
X'01'	Cumulative check one (CRC SDLC)
X'02'	Cumulative check two (LRC)

The second operand address points to a halfword residual checksum to be included in the cumulative check.

Operation

If CRC BISYNC is specified, the data byte, and the old residual checksum participate in the Generation of a new residual checksum based on the evaluation of the polynomial $(x^{16} + x^{15} + x^{2} + 1)$.

If CRC SDLC is specified, a similar operation is performed, using the polynomial $(x^{16} + x^{12} + x^{5} + 1)$

In both of these cases, the new residual checksum replaces the old residual checksum at the second operand location.

If LRC is specified, the EXCLUSIVE OR of the data byte with the old residual checksum replaces the old residual checksum at the second operand location.

Condition Code

Unchanged

Programming Note

Bits 0:7 and 16:23 of the register specified by R1 are ignored.

The register specified by R1 remains unchanged.

The second operand must be located on a halfword boundary.

Undefined check codes should not be used. If they are, the results are undefined.

Example: PB

This example performs a Process Byte instruction and stores the residue into RESIDUE.

Register 1	contains X'0001007A'
where:	01 = CRC SDLC
	7A = DATA BYTE
RESIDUE	contains X'D053' = old residue

Assembler Notation	Comments
PB R1, RESIDUE	RESIDUE ON HALFWORD BOUNDARY

Result of PB Instruction

(R1)	= unchanged by this instruction
(RESIDUE)	= X'BC13' = new residue
Condition Code	= unchanged by this instruction

INSTRUCTION

Process Byte Register (PBR)

Assembler Notation	Op-Code	Format
PBR R1, R2	32	RR

Set Up

	0 7	8 15	16 23	24 31
R1	X	CHECK CODE	X	DATA BYTE
R2	0		RESIDUAL CHECKSUM	

Bits 24:31 of the register specified by R1 contain the data byte to be processed. Bits 8:15 of the register specified by R1 contain a check code to indicate the type of processing. This byte is interpreted as follows:

X'00'	Cumulative check zero (CRC BISYNC)
X'01'	Cumulative check one (CRC SDLC)
X'02'	Cumulative check two (LRC)

The second operand is a fullword contained in the register specified by R2. Bits 16:31 of the second operand contain the residual checksum to be included in the processing.

Operation

If CRC BISYNC is specified, the data byte, and the old residual checksum participate in the generation of a new residual checksum based on the evaluation of the polynomial ($x^{16} + x^{15} + x^2 + 1$).

IF CRC SDLC is specified, a similar operation is performed, using the polynomial ($x^{16} + x^{12} + x^5 + 1$).

In both these cases, the new residual checksum replaces the contents of the Bits 16:31 of the register specified by R2.

If LRC is specified, the EXCLUSIVE OR of the data byte with the old residual checksum replaces the old residual checksum in the second operand.

Condition Code

Unchanged

Programming Note

Bits 0:7 and 16:23 of the register specified by R1 are ignored. The register specified by R1 remains unchanged. Bits 0:15 of the register specified by R2 are not used and must be zero.

Undefined check codes should not be used. If they are, the results are undefined.

INSTRUCTION

Move and Process Byte String Register (MPBSR)

Assembler Notation	Op-Code	Format
MPBSR R1, R2	30	RR

Set Up

	0 7	8 15	16 31
R1	DATA BYTES STRING ADDRESS		
R1+1	TRANSLATION TABLE ADDRESS		
R1+2	CONTROL CODE	CHECK CODE	COUNT
R1+3	0		RESIDUAL CHECKSUM
R1+4	LINK REGISTER FOR SUBROUTINE		
R2	DESTINATION BUFFER ADDRESS		

The register specified by R1 contains the address of the first byte in the string to be moved and processed.

The register specified by R1+1 contains the address of the translation table.

Bits 0:7 of the register specified by R1+2 contain a control code to indicate both the type and the sequence of processing. This byte is defined as follows:

X'00'	Cumulative check using data byte, move data byte
X'08'	Translate, cumulative check using data byte, move translated byte
X'0A'	Translate, cumulative check using translated byte, move translated byte
X'0C'	Translate, move translated byte

Bits 8:15 of the register specified by R1+2 contain a check code to indicate the type of cumulative check to be used in processing the data bytes. This byte is interpreted as follows:

X'00'	Cumulative check zero (CRC BISYNC)
X'01'	Cumulative check one (CRC SDLC)
X'02'	Cumulative check two (LRC)

If cumulative check is not specified, this byte does not participate in the MPBSR instruction.

Bits 16:31 of the register specified by R1+2 contain a halfword count which defines the number of bytes to be processed. A count of X'0000' specifies a move of 1 character. A count of X'7FFF' specifies a move of 32,768 characters. These are the minimum and maximum values respectively.

Bits 16:31 of the register specified by R1+3 contain the halfword residual value to be used in performing the cumulative check. If cumulative check is not specified, this register does not participate in the MPBSR instruction.

The register specified by R1+4 is used as a link register in the translation process, if a special character subroutine is specified. If translate is not specified or if a special character routine is not specified, this register does not participate in the MPBSR instruction.

The register specified by R2 contains the address of the destination buffer.

Operation

Refer to Figure 10-1.

Successive bytes, starting with the first in the source string are:

1. Processed in accordance with the specified codes.
2. Moved to the destination buffer.

The operation stops when the byte count becomes negative. The source string is unchanged. (See Addresses and Count, below.) The processed bytes replace the contents of the destination buffer. Upon completion of the instruction, the location counter is incremented to point to the next instruction in sequence. If the byte count is negative at the start of the instruction, no moving or processing is done, the instruction terminates, and the location counter is incremented to point to the next instruction.

Translation

The translation operation requires a 256 halfword table located in memory at the address contained in the register specified by R1+1. The table is arranged in ascending order, with one entry for each of the 256 possible data bytes. The translation operation may result in either a direct replacement, (in the destination buffer), of the data byte with another, or in a transfer to a special character subroutine.

If the most significant bit, bit zero, of the halfword entry corresponding to the data byte is a one, then bits 8:15 contain the replacement byte. This byte is moved to the proper location in the destination buffer. The table entry is unchanged.

If the most significant bit of the entry is a zero, then bits 1:15 contain the address, divided by two, of the special character subroutine. Before transferring to the subroutine, the link register, specified by R1+4, is loaded with the address of the MPBSR instruction. The source address has not been incremented and points to the current byte. The count has not been decremented. The destination address has not been incremented and points to the proper destination for this byte. This byte does not participate in the cumulative check.

If none of the halfwords in the translation table has its most significant bit set (i.e., no special character subroutines), the register specified by R1+4 is not used by this instruction.

Cumulative Check

The source byte used for the cumulative check may be the data byte or the translated byte as specified by the control code. The source byte is included in any one of three types of cumulative check operations as specified by the check code.

If CRC BISYNC is specified, the source byte, and the old residual checksum contained in Bits 16:31 of the register specified by R1+3 participate in the generation of a new residual checksum using a cyclic redundancy checking algorithm based on the generated polynomial $(x^{16} + x^{15} + x^2 + 1)$.

If CRC SDLC is specified, a similar operation is performed, using the polynomial $(x^{16} + x^{12} + x^5 + 1)$.

In both of these cases, the new residual checksum replaces the contents of Bits 16:31 of the register specified by R1+3.

If LRC is specified, the EXCLUSIVE OR of the source byte with the old residual checksum replaces the old residual checksum in Bits 16:31 of the register specified by R1+3.

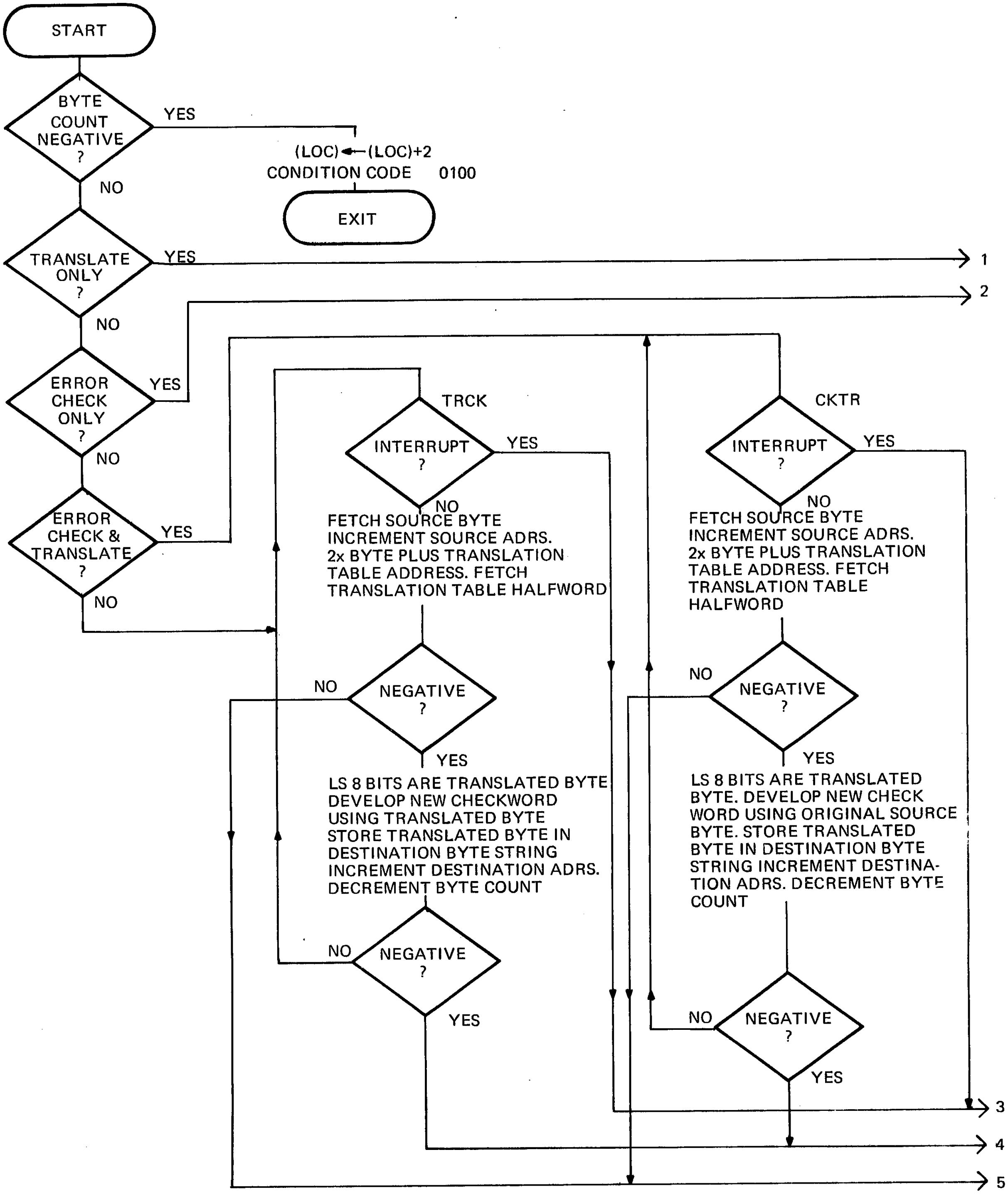

START
BYTE COUNT NEGATIVE ?
YES
NO
(LOC)←(LOC)+2
CONDITION CODE 0100
EXIT
TRANSLATE ONLY ?
YES
NO
1
2
ERROR CHECK ONLY ?
YES
NO
ERROR CHECK & TRANSLATE ?
YES
NO
TRCK
INTERRUPT ?
YES
NO
FETCH SOURCE BYTE INCREMENT SOURCE ADRS. 2x BYTE PLUS TRANSLATION TABLE ADDRESS. FETCH TRANSLATION TABLE HALFWORD
NEGATIVE ?
NO
YES
LS 8 BITS ARE TRANSLATED BYTE DEVELOP NEW CHECKWORD USING TRANSLATED BYTE STORE TRANSLATED BYTE IN DESTINATION BYTE STRING INCREMENT DESTINATION ADRS. DECREMENT BYTE COUNT
NEGATIVE ?
NO
YES
CKTR
INTERRUPT ?
YES
NO
FETCH SOURCE BYTE INCREMENT SOURCE ADRS. 2x BYTE PLUS TRANSLATION TABLE ADDRESS. FETCH TRANSLATION TABLE HALFWORD
NEGATIVE ?
NO
YES
LS 8 BITS ARE TRANSLATED BYTE. DEVELOP NEW CHECK WORD USING ORIGINAL SOURCE BYTE. STORE TRANSLATED BYTE IN DESTINATION BYTE STRING INCREMENT DESTINA-TION ADRS. DECREMENT BYTE COUNT
NEGATIVE ?
NO
YES
3
4
5

Figure 9-1. Flowchart of MPBSR Instruction (Sheet 1 of 2)

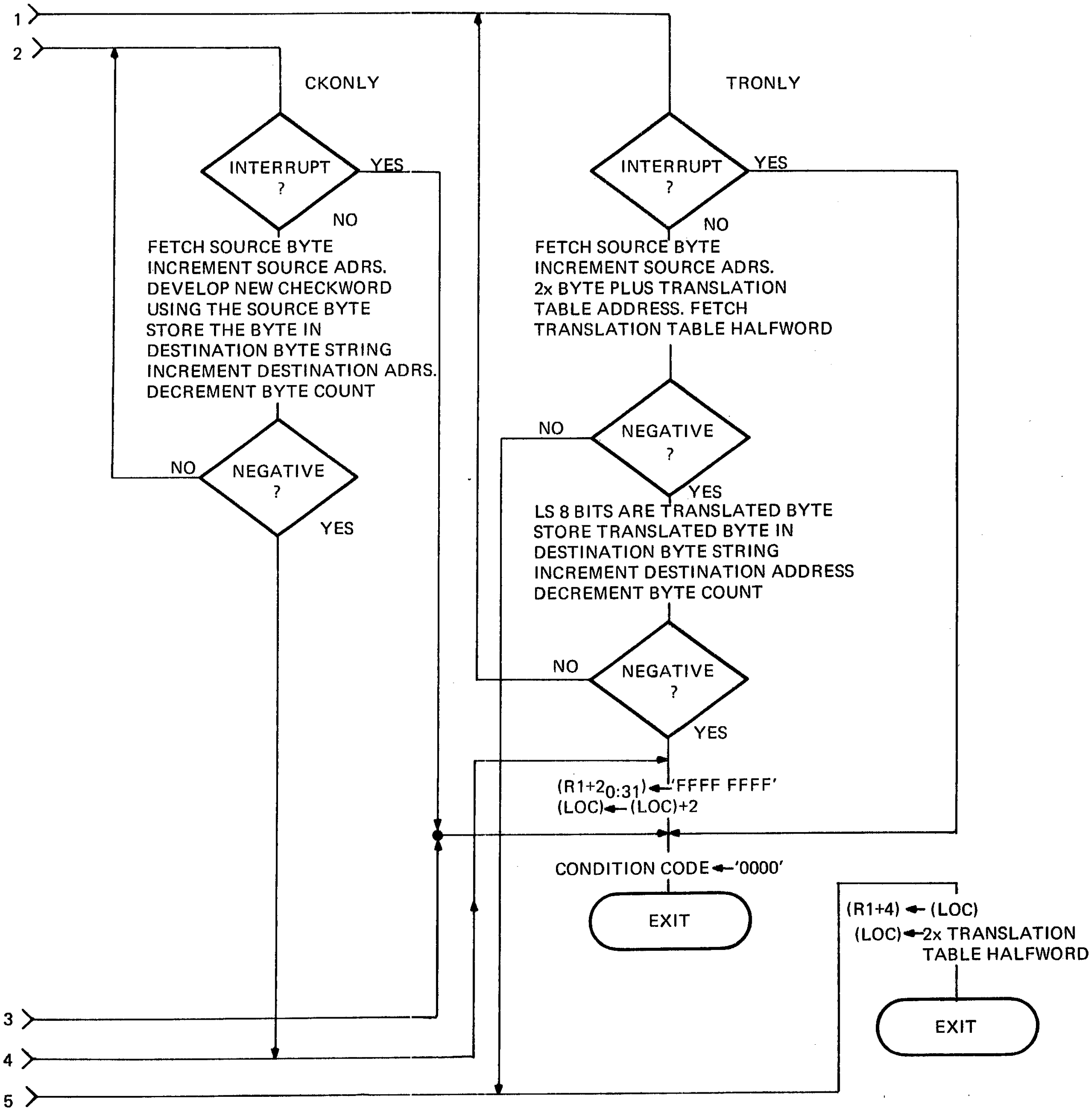


Figure 9-1. Flowchart of MPBSR Instruction (Sheet 2 of 2)

Byte Count

As each byte is moved, the source address and the destination address are incremented by one. The count is decremented by one. Upon completion of the instruction, the source and destination address registers contain the incremented addresses. The count register specified by R1+2 contains a negative one, X'FFFF FFFF'.

The count value is equal to the number of bytes in the source string minus one. A count of X'0000' causes one byte to be processed, a count of X'7FFF' causes 32,768 bytes to be processed. These are the minimum and maximum count values respectively.

Condition Code

C	V	G	L	
0	0	0	0	Successful completion
0	1	0	0	Count negative at start

Addresses

There are no boundary restrictions on either the location of the source string or on the location of the destination buffer. Either may start and end on odd byte boundaries. If the memory access controller is present and enabled, memory references using these addresses are relocated.

The translation table must be located on a halfword boundary. The address of the translation table is relocated, if the memory access controller is present and enabled. Within the translation table, the address of the special subroutine must point to a location within the first 64KB of program space. This address is also subject to relocation by the memory access controller.

Source and destination buffers may overlap. No checking is performed. The addresses specified by the source (R1) and destination (R2) registers may be equal, specifying a move in place, but R1 must not be equal to R2. That is, the instruction MPBSR 3,3 is invalid.

Programming Note

This instruction is interruptable. The point at which interrupts are recognized, and the periods of non-interruptability may vary in different implementations. Any of the following events may cause this instruction to be interrupted: machine malfunction, memory failure, memory access violation, external device attention. Before taking the interrupt, the processor finishes processing the current byte, increments the source and destination addresses, and decrements the count. The location counter is not incremented. This permits the move to resume, following the servicing of the interrupt. Interrupt routines may use this instruction, provided they do not destroy the contents of the registers.

Undefined control codes should not be used. If they are, the results are unpredictable.

Illegal instruction interrupt occurs if the Processor is not equipped with the communication Instructions option.

If R1 specifies register number 6, then registers 6, 7, 8, 9 and 10 are used by this instruction. If R1 specifies register number 13, then registers 13, 14, 15, 0 and 1 are used, in that order, by this instruction.

If R1 = R2, the results are not defined.

EXAMPLE: MPBSR

This example moves and performs a CRC SDLC check on a byte string of data.

BUFIN = 256 bytes buffer containing data 0:X'FF'

Register 1	contains address of BUFIN
Register 2	contains address of TRANSTAB
Register 3	contains X'000100FF' where: 00 indicates check and move 01 indicates CRC SDLC 00 is not used FF indicates 256 bytes to be used
Register 4	contains X'0' to begin
Register 5	not used in this example
Register 6	contains address of BUFOUT

Assembler Notation		Comment
MPBSR	REG1, REG6	MOVE BUFIN TO BUFOUT

(REG1)	=	BUFIN + 256
(REG2)	=	unchanged by this instruction
(REG3)	=	X'FFFF FFFF'
(REG4)	=	Half Residue X'D841'
(REG6)	=	BUFOUT + 256
Condition Code	=	0000 successful completion

BUFIN is unchanged
BUFOUT now contains 256 bytes 0-255

CHAPTER 10

M71-102 HEXADECIMAL DISPLAY PANEL AND M71-101 BINARY DISPLAY PANEL PROGRAMMING SPECIFICATION

INTRODUCTION

The M71-102 Hexadecimal Display Panel and M71-101 Binary Display Panel provide a means to manually control the Processor, interrogate and display various Processor registers and machine status, set and display Processor memory locations, and may be programmed as an I/O device by the user. The Hexadecimal Display Panel and Binary Display Panel are identical in operation. For convenience of the operator the Hexadecimal Display is equipped with a Hexadecimal readout in addition to the standard Binary readout.

CONFIGURATION

The Hexadecimal Display Panel provides the system operator with visual indications of the state of the Processor, as well as manual control over the system.

The Hexadecimal Display Panel, shown in Figure 10-1, is a RETMA standard $5\frac{1}{4}$" x 19" panel which is plug removable from the Processor. It displays the current state of the Processor and provid·s all necessary manual control over the system. The following paragraphs describe the control and display elements of the Hexadecimal Display Panel.

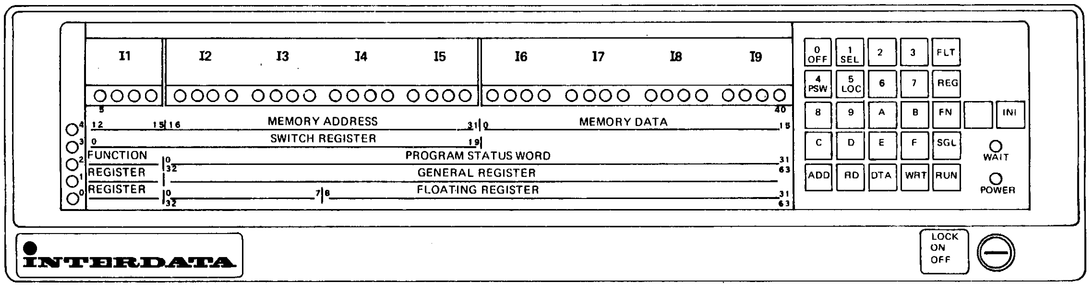


Figure 10-1. Hexadecimal Display Panel

Display Registers and Indicators

Internal to the Hexadecimal Display Panel are five eight-bit byte Display Registers, D1 through D5, that hold data output from the Processor, and a 20-bit Switch Register that holds data input from The Hexadecimal Keyboard. Refer to Figure 10-2.

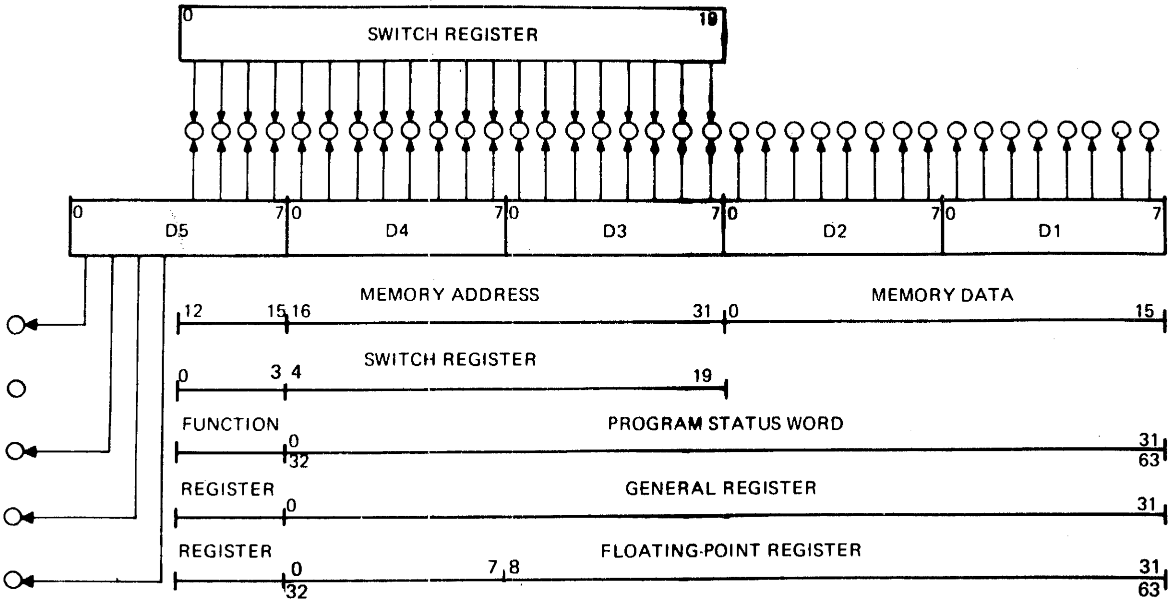


Figure 10-2. Display Registers and Indicators

Associated with each of Display Registers D1 through D4 are eight indicator lamps that provide a binary read-out and two optional hexadecimal read-out indicators. Associated with the least significant four bits of Display Register D5 are four indicator lamps for binary display and one optional hexadecimal read-out indicator.

The most significant four bits of Display Register D5 (Bits 0:3) control four of the five indicator lamps along the left edge of the Hexadecimal Display Panel. The fifth indicator lamp is controlled by logic internal to the Hexadecimal Display Panel. To the right of each of these five lamps is a diagram that defines what is being displayed. In general, only one of the diagram lamps is on at a time. If none of the diagram lamps are on, a user program has written data to the Display Register D5.

As seen in Figure 10-2, the most significant 20-bits of the display show the contents of Display Registers D3 and D4 and the least significant four bits of Display Register D5 (Bits 4:7); or the contents of the 20-bit Switch Register. When the Switch Register is being displayed, the lamp next to the Switch Register diagram is turned ON. Any other diagram lamp that may have been ON, remains ON. When the Switch Register is no longer displayed, its diagram lamp goes out and the most significant 20-bits of the display again show the contents of Display Registers D3 and D4 and the least significant four bits of Display Register D5 (Bits 4:7).

The methods of displaying the Switch Register and the other diagrammed items are discussed later.

Key Operated Security Lock

This is a three-position, OFF-ON-LOCK, key-operated locking switch, which controls the primary power to the system. This switch can also disable the Hexadecimal Display Panel, thereby preventing any accidental manual input to the system. The power indicator lamp (PWR) associated with the key lock is located in the lower right corner of the Hexadecimal Display Panel. The PWR lamp is ON when the key lock is in the ON or LOCK position. The relationship between the key lock switch positions, primary power, the Control keys, and the Hexadecimal keys is:

OFF — The primary power is OFF.

ON — The primary power is ON and the Control keys and Hexadecimal keys are enabled.

LOCK — The primary power is ON and the Control keys and Hexadecimal keys are disabled. Only INT switch is active.

Control Keys

The momentary contact Control keys are only active when the key-operated locking switch is in the ON position.

INITIALIZE (INT) — The Initialize (INT) key causes the system to be initialized. After the initialize operation, all device controllers on the system Multiplexor Bus are cleared and certain other functions in the Processor are reset.

DATA (DTA) — The Data (DTA) key clears the Switch Register and connects it to the most significant 20 display indicators. The Switch Register diagram lamp is turned ON. Hexadecimal data may now be entered into the Switch Register from the Hexadecimal Keyboard. As each Hexadecimal key is depressed, the data shifts into the Switch Register from the right. If more than five hexadecimal digits are entered, data shifted out of the Switch Register is lost.

Depressing any non-hexadecimal key disconnects the Switch Register from the high order 20 display lamps and extinguishes the Switch Register diagram lamp.

ADDRESS (ADD) — The Address (ADD) key causes the Processor to halt and copy the contents of the Switch Register into the Location Counter field of the Program Status Word. The new value of the Location Counter is then output to Display Registers D1, D2, D3, and D4. The function diagram lamp is turned ON and a Hexadecimal 5 is output to the top four display lamps (Bits 4:7 of D5).

MEMORY READ (RD) — The Memory Read (RD) key causes the Processor to halt and read the halfword contents of the memory location presently pointed to by the Location Counter. (If the Memory Access Controller is enabled then the relocated value of the Location Counter is the effective address of the memory location.) The halfword data read is output to Display Registers D1 and D2. The Location Counter is incremented by two and output to Display Registers D3 and D4 and the least significant four bits of Display Register D5 (a 20-bit value). The lamp next to the Memory Address/Memory Data diagram is turned ON.

MEMORY WRITE (WRT)

The Memory Write (WRT) key causes the Processor to halt and read in the least significant 16 bits of the 20 bit Switch Register. The halfword of data is written into the memory location presently pointed to by the Location Counter. (If the Memory Access Controller is enabled then the relocated value of the Location Counter is the effective address of the memory location.) The data written is then output to Display Registers D1 and D2. The Location Counter is incremented by two and output to Display Registers D3 and D4 and the least significant four bits of Display Register D5. The lamp next to the Memory Address/Memory Data diagram is turned ON.

EXAMINE REGISTER (REG)

The Examine Register (REG) key sets up the Hexadecimal Display Panel to interpret the next Hexadecimal key depressed as a General Register number. When the hexadecimal register number key is depressed, the Processor halts and the content of the selected General Register of the current register set is output to Display Registers D1, D2, D3 and D4. The General Register diagram lamp is turned ON and the number of the displayed register is output to the top four display lamps.

EXAMINE FLOATING-POINT REGISTER (FLT)

The Examine Floating-Point Register (FLT) key sets up the Hexadecimal Display Panel to interpret the next hexadecimal key depressed as the number of a Floating-Point Register. When the hexadecimal register number key is depressed, the Processor halts and the content of the selected Floating-Point Register is output to Display Registers D1, D2, D3, and D4. The Floating-Point Register diagram lamp is turned ON and the number of the displayed register is output to the top four display lamps. If an odd numbered register had been selected and the processor is not equipped with double precision option, the register number is forced to the next lower even value before being used. On Processors not equipped with floating-point, the result of this operation is undefined.

FUNCTION (FN)

The Function (FN) key sets up the Hexadecimal Display Panel to interpret the next hexadecimal key depressed as the number of one of sixteen functions. When the hexadecimal key is depressed, the Processor halts to interpret the selected function. If the function is undefined, the Processor remains halted with no change to the display indicators. The defined functions are detailed later.

SINGLE STEP (SGL)

The Single Step (SGL) key causes the Processor to execute one user level instruction at current location counter, increment the LOC and then halt. The register that was selected (PSW, LOC, General Register, etc.) is displayed.

RUN (RUN)

The Run (RUN) key causes the Processor to begin program execution at the address pointed to by the Location Counter (LOC).

OPERATING PROCEDURES

Power Up

To power up the system, turn the key-operated security lock clockwise from the OFF position to the ON position. This action provides electrical power to the system and leaves all device controllers on the Multiplexor Bus in an initialized state.

Power Down

To shut down power to the system:

1. Halt the Processor.
2. Turn the key-operated security lock to the OFF position.

This removes primary power from the system and forces a Primary Power Fail (PPF) interrupt to the Processor. When power is re-applied, the Processor displays the contents of the Location Counter (LOC) and then assumes the Halt mode. If the Processor had been running when power was turned OFF, the Run mode is assumed when power is re-applied.

Address a Memory Location

To select an address:

1. Depress the Data (DTA) key. The Switch Register is cleared and displayed.
2. Enter the desired address from the Hexadecimal Keyboard.
3. Depress the Address (ADD) key. The Processor halts and copies the contents of the Switch Register into the Location Counter field of the PSW. The new value of the Location Counter is then displayed.

Memory Read

To display the contents of memory locations:

1. Select the memory read start address as in Address a Memory Location.
2. Depress the Read (RD) key. The address read from, plus two, and the data read from memory are displayed.
3. Repeat from Step 2 to read successive memory locations. The Location Counter is automatically incremented by two each time RD is depressed.

Memory Write

To write data from the Switch Register into memory:

1. Select the memory write start address as in Address a Memory Location.
2. Depress the Data (DTA) key. The Switch Register is cleared and displayed.
3. Enter the data to be written from the Hexadecimal Keyboard.
4. Depress the Write (WRT) key. The address written into, plus two, and the data written are displayed.
5. Repeat from Step 2 to write different data into successive locations or from Step 4 to write the same data into successive locations. The Location Counter is automatically incremented by two each time WRT is depressed.

General Register Display

To examine the contents of a General Register:

1. Depress the Register (REG) key.

2. Depress the hexadecimal register number. The Processor halts and the contents of the selected General Register is displayed.

NOTE

> The General Register displayed is from the register set specified by the current Program Status Word.

Floating-Point Register Display

To examine the contents of a Floating-Point Register:

1. Depress the Floating-Point Register (FLT) key.

2. Depress the hexadecimal register number. If the Processor is not equipped with floating-point the result of this operation is undefined. If the Processor is equipped with floating-point, the selected register number is forced even and the Floating-Point Register is displayed. The Processor is left in the Halt mode.

Floating-Point Register Display (later versions of 7/32)

After initialize or after a Function 2 all manual references to floating register are single precision. After a Function 3 all references to floating registers are double precision, if the Double Floating Point Unit (DFU) is equipped.

Using even/odd concept

The even numbered register of an even/odd pair refers to the most significant 32 bits and the odd numbered register refers to the least significant 32 bits.

References to an odd numbered floating point register when in the single precision mode (FN 2) produce different results depending on whether or not the DFU is equipped. If DFU is absent then the number is forced to the next lower even number and that single precision register is displayed. If DFU is present then the LS 32 bits of the corresponding double register are displayed.

Program Status Word Display and Modification

To examine the Status field (most significant half) of the current PSW:

1. Depress the Function (FN) key.

2. Depress Hexadecimal key 4. The Processor halts and the status field (most significant half) of PSW is displayed.

To examine the Location Counter field (least significant half) of the current PSW:

1. Depress the Function (FN) key.

2. Depress Hexadecimal key 5. The Processor halts and the Location Counter field (least significant half) of PSW is displayed.

To modify the least significant 16 bits (Bits 16-31) of the Status field:

1. Depress the Data (DATA) key.

2. Enter the data (to be written into bits 16-31 of the PSW) from the Hexadecimal keyboard.

3. Depress the Function (FN) key.

4. Depress Hexadecimal key 1. The Processor halts and copies the 16 bits of the Switch register in bits 16-31 of the PSW. The modified PSW is then displayed.

Program Execution

To begin execution of a program:

1. Select the program start address as in Address a Memory Location.

2. Select the register to be displayed.

3. Depress the Run (RUN) key.

To execute a program in the Single-Step mode:

1. Select the program start address as in Address a Memory Location.

2. Select the register to be displayed.

3. Depress the Single-Step (SGL) key. One instruction is executed, the last selected register (PSW, LOC, General Register, etc.) is displayed and the Processor halts.

4. Repeat Step 3 to execute successive instructions. Return to Step 2 to display different registers.

Program Termination

To manually halt the execution of a program, display any register or depress the Single-Step (SGL) key. In the latter case, the last selected register is displayed.

Console Interrupt

To generate an interrupt from the Hexadecimal Display Panel:

1. Depress the Function (FN) key.

2. Depress Hexadecimal key 0. If enabled by the current PSW, an interrupt from device number 1 is simulated. If not enabled, the Processor enters the Run mode. Hexadecimal Display Panel interrupts are not queued.

The Hexadecimal Display Panel interrupt feature allows an operator to inform the running program that some operator service or function is needed. No acknowledgement of the interrupt is required of the running program.

Switch Register

To examine the Switch Register at any time during execution of a program, depress any hexadecimal key. The Switch Register is displayed for as long as the key is depressed. No information enters the Switch Register. When the hexadecimal key is released, the top 20 display lamps return to their previous state.

The Switch Register can be modified without interrupting the Processor as follows:

1. Depress the Data (DTA) key. The Switch Register is cleared and displayed.

2. Enter the desired hexadecimal data.

Power Fail

When the Processor detects a power failure, the micro-program senses the Hexadecimal Display Panel status. The present status of the display is stored in main memory at a dedicated area by the micro-program. The current Program Status Word, Location Counter and the programmable registers are then saved in dedicated main memory locations and the micro-program deactivates the System Clear (SCLR) relay.

On power up, after the system clear relay has re-activated, the Program Status Word, Location Counter, and programmable registers are restored from their main memory save locations. The status of the display prior to the power failure is retrieved and interrogated by the micro-program.

If the Hexadecimal Display Panel was in the Run mode, and the Initialize Key is not depressed, and if the Machine Malfunction Interrupt Enable bit of the PSW is set, a Machine Malfunction Interrupt is taken. If Machine Malfunction Interrupts are not enabled, the Processor enters the Run mode beginning at the instruction pointed to by the Location Counter.

If the Hexadecimal Display Panel was not in the Run mode, or if the Initialize Key is still depressed, the value of the Location Counter is output to the display registers, the WAIT lamp on the console is turned ON and the Halt mode is entered.

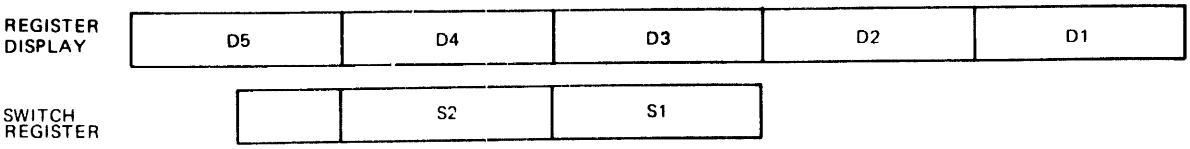
Power failure and operation of the Initialize key are indistinguishable to the Micro-Program except as described above. Consequently, operation of the Initialize key should be considered carefully when the Machine Malfunction Interrupt is enabled. The Initialize Key causes all the activities associated with a power failure to occur. The System Clear relay deactivates, then, after some delay, it is re-activated. If, after these electro-mechanical delays, the Initialize Key is still being depressed, the Halt mode is entered. The total delay works out to be about a half a second.

Care should also be taken when using the Hexadecimal Display Panel as an input device (testing Switch Register bits) due to the volatility of the Switch Register in a power fail situation.

After a power up, the contents of the Switch Register are undefined. The display status byte is forced to X'40' on power up or initialize.

DATA FORMAT

A byte or a halfword can be transferred to or from the Display using the WD, WH, WDR, WHR, or RD, RH, RDR, RHR instructions. Refer to Figure 10-3.



	DATA TRANSFERRED	
INSTRUCTIONS EXECUTED	NORMAL MODE	INCREMENTAL MODE
RD (R)	S1	S1
RD (R)	S1	S2
RD (R)	S1	S1
RD (R)	S1	S2
RH (R)	S1,S2	S1,S2
RB (R) *	S1,S2,S1.S2	S1,S2,S1,S2
WD (R)	D1	D1
WD (R)	D1	D2
WD (R)	D1	D3
WD (R)	D1	D4
WD (R)	D1	D5
WH (R)	D1,D2	D1,D2
WH (R)	D1,D2	D3,D4
WH (R)	D1,D2	D5,NOTE 1
WB (R) * *	D1,D2,D3,D4,D5	D1,D2,D3,D4,D5

* BLOCK LENGTH = 4 BYTES   ** BLOCK LENGTH = 5 BYTES   NOTE 1. SUBSEQUENT BYTES OUTPUT ARE LOST.

Figure 10-3. Hexadecimal Display Panel Data Transfers

PROGRAMMING INSTRUCTIONS

Input/Output Programming

The Hexadecimal Display Panel is available to any running program as an I/O device with device address 01. The status and command bytes for the Hexadecimal Display Panel are summarized in Table 10-1. The status byte indicates the mode of the Hexadecimal Display Panel and is of little interest to a running program as it always indicates Run mode or Hexadecimal Display Panel Interrupt (Function 0). The command byte selects Normal or Incremental mode, which pertains to data Transfers. The selection logic which determines the Switch Register byte or register display byte to transfer is reset every time the Hexadecimal Display Panel is addressed when in the Normal mode. When an Output Command Incremental mode is issued to the Hexadecimal Display Panel, the byte selection logic is initially reset. Subsequent Read or Write instructions transfer bytes as shown in Figure 10-3.

Block I/O with the Hexadecimal Display Panel is only feasible when the least significant four status bits are reset.

NOTE

> After an initialize sequence or after any manual Hexadecimal Display Panel operation that results in anything being displayed, the Display Device Controller is automatically placed in the Normal mode.

When programming the Hexadecimal Display Panel in the Incremental mode, the Output Command Incremental mode must be issued before each set of data transfers to guarantee that the byte selection logic is reset.

The most significant four bits of the Switch Register are only available to the micro-program. These four bits are transferred as Bits 5, 6, 7, and 0 of the status when the Hexadecimal Display Panel status is Address (i.e., Display Status = X'011XXXX').

Wait State

The running program can place the Processor into the Wait state by setting the Wait bit of the current PSW. The WAIT indicator on the lower right of the panel is turned ON to inform the operator of the Wait state. The Processor can leave the Wait state and resume execution in two ways:

1. An Interrupt can occur causing the Processor to jump to an interrupt service routine. When the routine restores the original PSW, the Wait state is re-established.

2. The operator can depress the RUN key which causes the Wait bit in the PSW and the WAIT lamp to be reset and execution to resume at the address specified by LOC.

PROGRAMMING SEQUENCES

The Hexadecimal Display has a device address of X'01'.

This device can be used to output up to five bytes of data to the Console Panel Indicators. The following program sequence outputs four bytes of data starting from the memory location BUF:

```
LIS     R1,1        (R1) = Display Address
LHI     R3,X'40'
OCR     R1,R3        Display in Incremental Mode
WD      R1,BUF
WD      R1,BUF+1
WD      R1,BUF+2
WD      R1,BUF+3
```

At this time the Console Panel Indicators are ON as shown below:

D5	D4	D3	D2	D1
	(BUF+3)	(BUF+2)	(BUF+1)	(BUF)

In order to light indicators D1 and D2, the Console can be in the normal mode and one halfword can be output. The following programming sequenoe can be used:

```
LIS      R1,1
LHI      R3,X'80'
OCR      R1,R3        Console in Normal Mode
WH       R1,BUF
```

The Console Panel Indicators will be ON as shown below:

D5	D4	D3	D2	D1
			(BUF+1)	(BUF)

Note that when a halfword of data is output to the Console Panel, the most significant byte loads in indicator D1 and the least significant byte loads in D2.

The Console Panel Switch Register can be read by using the read instructions as shown below:

```
LIS      R1,1         (R1) = Console Address
LHI      R3,X'80'     (R3) = 80 = Normal Mode
OCR      R1,R3
RHR      R1,R4        Read 1 Halfword
EXBR     R4,R4        Exchange Bytes
```

At this time Register 4 has the 16 data switches.

Programming Note:

If more than five bytes are output to the Display Panel, the data is lost after five bytes. The Console must then be initialized by giving an Output Command to it before outputting any data, if the data is to be displayed.

TABLE 10-1. DISPLAY STATUS AND COMMAND

STATUS

	0	1	2	3	4	5	6	7
Run	X	0	0	0	X	X	X	X
Memory write	X	0	0	1	X	X	X	X
Memory read	X	0	1	0	X	X	X	X
Address	X	0	1	1	X	X	X	X
Fixed Register	X	1	0	0	X	X	X	X
Floating Register	X	1	0	1	X	X	X	X
Function	X	1	0	0	X	X	X	X

General Register 0	0	1	0	X	1	0	0	0	Floating Register 0
1	1	1	0	X	1	0	0	0	
2	0	1	0	X	1	0	0	1	2
3	1	1	0	X	1	0	0	1	
4	0	1	0	X	1	0	1	0	4
5	1	1	0	X	1	0	1	0	
6	0	1	0	X	1	0	1	1	6
7	1	1	0	X	1	0	1	1	
8	0	1	0	X	1	1	0	0	8
9	1	1	0	X	1	1	0	0	
A	0	1	0	X	1	1	0	1	A
B	1	1	0	X	1	1	0	1	
C	0	1	0	X	1	1	1	0	C
D	1	1	0	X	1	1	1	0	
E	0	1	0	X	1	1	1	1	Floating Register E
General Register F	1	1	0	X	1	1	1	1	

Function 0	0	1	0	0	0	0	0	0	Console Interrupt
1	1	1	0	0	0	0	0	0	PSW Select
2	0	1	0	0	0	0	0	1	Set Single precision display mode
3	1	1	0	0	0	0	0	1	Set Double precision display mode
4	0	1	0	0	0	0	1	0	Display PSW
5	1	1	0	0	0	0	1	0	Display LOC
6	0	1	0	0	0	0	1	1	
7	1	1	0	0	0	0	1	1	
8	0	1	0	0	0	1	0	0	
9	1	1	0	0	0	1	0	0	
A	0	1	0	0	0	1	0	1	
B	1	1	0	0	0	1	0	1	
C	0	1	0	0	0	1	1	0	
D	1	1	0	0	0	1	1	0	
E	0	1	0	0	0	1	1	1	
Function F	1	1	0	0	0	1	1	1	

COMMAND

Normal	1	0	0	0	0	0	0	0
Incremental	0	1	0	0	0	0	0	0

APPENDIX 1

MODEL 7/32 OP-CODE MAP

LSD ↓ / MSD →	0	1	2	3	4	5	6	7	9	C	D	E	F
0		SRLS	BTBS	MPBSR[1]	STH	ST[4]	STE[2][4]	STD[3][4]	SRHLS	BXH	STM[4]	TS	
1	BALR	SLLS	BTFS		BAL	AM[4]	AHM	STME[2][4]	SLHLS	BXLE	LM[4]	SVC[4]	
2	BTCR	CHVR	BFBS	PBR[1]	BTC		PB[1]	LME[2][4]	STBR	*LPSW	STB	*SINT	
3	BFCR		BFFS		BFC		LRA[4]	LHL	LBR	THI	LB	*SCP[4]	TI
4	NR		LIS	EXHR	NH	N[4]	ATL[4]	TBT	EXBR	NHI	CLB		NI
5	CLR		LCS		CLH	CL[4]	ABL[4]	SBT	*EPSR	CLHI	*AL		CLI
6	OR		AIS		OH	O[4]	RTL[4]	RBT	*WBR	OHI	*WB[4]	LA	OI
7	XR		SIS		XH	X[4]	RBL[4]	CBT	*RBR	XHI	*RB[4]	TLATE[4]	XI
8	LR	*LPSWR	LER[2]	LDR[3]	LH	L[4]	LE[2][4]	LD[3][4]	*WHR	LHI	*WH		LI
9	CR		CER[2]	CDR[3]	CH	C[4]	CE[2][4]	CD[3][4]	*RHR	CHI	*RH		CI
A	AR		AER[2]	ADR[3]	AH	A[4]	AE[2][4]	AD[3][4]	*WDR	AHI	*WD	RRL	AI
B	SR		SER[2]	SDR[3]	SH	S[4]	SE[2][4]	SD[3][4]	*RDR	SHI	*RD	RLL	SI
C	MHR	MR	MER[2]	MDR[3]	MH	M[4]	ME[2][4]	MD[3][4]		SRHL		SRL	
D	DHR	DR	DER[2]	DDR[3]	DH	D[4]	DE[2][4]	DD[3][4]	*SSR	SLHL	*SS	SLL	
E			FXR[2]	FXDR[3]		CRC12		STMD[3][4]	*OCR	SRHA	*OC	SRA	
F			FLR[2]	FLDR[3]		CRC16		LMD[3][4]		SLHA		SLA	

NOTES

1. Communication (Optional) Instructions. (Models 7/32C and 7/32C II only)
2. Single Precision Floating Point (Optional) Instructions.
3. Double Precision Floating Point (Optional) Instructions.
4. Second operand must be aligned on a fullword boundary.

* Privileged Instructions

APPENDIX 1 (Continued)

This manual describes all of the features (standard and optional) for all of the versions of Model 7/32 except M73-025 and M73-026. Refer to *Model 7/32 Reference Manual,* Publication Number 29-399 for a description of these two versions.

The following table shows the standard and optional features of the current versions of Model 7/32. Note that the optional features may be included with the initial system or may be added later. Certain optional features are required for certain software products. For example, OS/32-MT requires the Memory Access Controller (M73-104). The corresponding software manuals list all such requirements.

For further information, refer to *INTERDATA Price List,* Publication Number 38-074.

Model	Standard Features	Optional Features
7/32C	M73-030: 750 ns 64KB Core Memory or M73-031: 1000 ns 64KB Core Memory Standard 132 instructions	M71-101 Binary Display Panel M71-102 Hexadecimal Display Panel M73-100 Power Fail Detection/Auto Restart M73-101 Floating Point Hardware (Single Precision Only) M73-103 Direct Memory Access Buffer M73-104 Memory Access and Protect Controller M73-105 Extended Memory Selector Channel M73-106 Local Memory Bank Interface M73-107 Processor Parity Control M73-111 Local Memory Bank Interface Chassis M73-112 High Speed Data Handling (includes Data Communication Instructions and Hardware CRC) Up to 1 MB of core memory (750 ns or 1000 ns)
7/32C II	M73-032: 750 ns 64KB Core Memory or M73-033: 1000 ns 64KB Core Memory Standard 132 instructions	M71-101 Binary Display Panel M71-102 Hexadecimal Display Panel M73-100 Power Fail Detection/Auto Restart M73-103 Direct Memory Access Buffer M73-104 Memory Access and Protect Controller M73-105 Extended Memory Selector Channel M73-106 Local Memory Bank Interface M73-107 Processor Parity Control M73-111 Local Memory Bank Interface Chassis M73-112 High Speed Data Handling (includes Data Communication Instructions and Hardware CRC) M73-034 Floating Point Processor (Both Single and Double Precision) Up to 1 MB of core memory (750 ns or 1000 ns)

APPENDIX 2
INSTRUCTION SUMMARY - ALPHABETICAL WITH ATTRIBUTES

Attributes

A:	arithmetic fault interrupt can occur
C:	Condition Code in the PSW is set to reflect the result
D:	second operand must be on double work boundary for consistent result
F:	second operand must be on fullword boundary for consistent result
H:	second operand must be on halfword boundary for consistent result
I:	illegal instruction interrupt can be initiated
IA:	immediate interrupt or Auto-Driver Channel can be initiated
P:	protect mode violation can occur
RP:	relocation/protection interrupt can occur

INSTRUCTION	OP-CODE	MNEMONIC	ATTRIBUTES	PAGE NO.
Add	5A	A	C, F	4-4
Add Double Precision Floating Point	7A	AD	C, D, A, I	5-29
Add Floating Point	6A	AE	C, F, A, I	5-14
Add Floating Point Register	2A	AER	C, A, I	5-14
Add Halfword	4A	AH	C, H	4-5
Add Halfword immediate	CA	AHI	C	4-5
Add Halfword to Memory	61	AHM	C, RP, H	4-7
Add Immediate	FA	AI	C	4-4
Add Immediate Short	26	AIS	C	4-4
Add Register	0A	AR	C	4-4
Add Register Double Precision Floating Point	3A	ADR	C, A, I	5-29
Add to Bottom of List	65	ABL	C, F, RP	2-45
Add to Memory	51	AM	C, F, RP	4-6
Add to Top of List	64	ATL	C, F, RP	2-45
AND	54	N	C, F	2-21
AND Halfword	44	NH	C, H	2-22
AND Halfword Immediate	C4	NHI	C	2-22
AND Immediate	F4	NI	C	2-21
AND Register	04	NR	C	2-21
Autoload	D5	AL	C, P	7-14
Branch and Link	41	BAL	H	3-5
Branch and Link Register	01	BALR		3-5
Branch on False Condition	43	BFC	H	3-4
Branch on False Condition Backward Short	22	BFBS		3-4
Branch on False Condition Forward Short	23	BFFS		3-4
Branch on False Condition Register	03	BFCR		3-4
Branch on Index High	C0	BXH	H	3-7
Branch on Index Low or Equal	C1	BXLE	H	3-6
Branch on True Condition	42	BTC	H	3-3
Branch on True Condition Backward Short	20	BTBS		3-3
Branch on True Condition Forward Short	21	BTFS		3-3
Branch on True Condition Register	02	BTCR		3-3

APPENDIX 2 (Continued)

INSTRUCTION	OP-CODE	MNEMONIC	ATTRIBUTES	PAGE NO.
Compare	59	C	C, F	4-10
Compare Double Precision Floating Point	79	CD	C, D, I	5-31
Compare Floating Point	69	CE	C, F, I	5-18
Compare Floating Point Register	29	CER	C, I	5-18
Compare Halfword	49	CH	C, H	4-11
Compare Halfword Immediate	C9	CHI	C	4-11
Compare Immediate	F9	CI	C	4-10
Compare Logical	55	CL	C, F	2-18
Compare Logical Byte	D4	CLB	C	2-19
Compare Logical Halfword	45	CLH	C, H	2-19
Compare Logical Halfword Immediate	C5	CLHI	C	2-19
Compare Logical Immediate	F5	CLI	C	2-18
Compare Logical Register	05	CLR	C	2-18
Compare Register	09	CR	C	4-10
Compare Register Double Precision Floating Point	39	CDR	C, I	5-31
Convert to Halfword Value Register	12	CHVR	C	4-22
Complement Bit	77	CBT	C, RP	2-38
Cyclic Redundancy Check Modulo 12	5E	CRC12	H, RP	2-40
Cyclic Redundancy Check Modulo 16	5F	CRC16	H, RP	2-40
Divide	5D	D	F, A	4-14
Divide Double Precision Floating Point	7D	DD	C, D, A, I	5-33
Divide Floating Point	6D	DE	C, F, A, I	5-21
Divide Floating Point Register	2D	DER	C, A, I	5-21
Divide Halfword	4D	DH	H, A	4-16
Divide Halfword Register	0D	DHR	A	4-16
Divide Register	1D	DR	A	4-14
Divide Register Double Precision Floating Point	3D	DDR	C, A, I	5-33
Exchange Byte Register	94	EXBR		2-12
Exchange Halfword Register	34	EXHR		2-13
Exchange Program Status Register	95	EPSR	C, P, IA	6-14
Exclusive OR	57	X	C, F	2-25
Exclusive OR Halfword	47	XH	C, H	2-26
Exclusive OR Halfword Immediate	C7	XHI	C	2-26
Exclusive OR Immediate	F7	XI	C	2-25
Exclusive OR Register	07	XR	C	2-25
Fix Register	2E	FXR	C, I	5-23
Fix Register Double Precision Floating Point	3E	FXDR	C, I	5-34
Float Register	2F	FLR	C, I	5-24
Float Register Double Precision	3F	FLDR	C, I	5-35
Load	58	L	C, F	2-5
Load Address	E6	LA		2-7
Load Byte	D3	LB		2-11
Load Byte Register	93	LBR		2-11
Load Complement Short	25	LCS	C	2-5
Load Double Precision Floating Point	78	LD	C, D, A, I	5-25
Load Floating Point	68	LE	C, F, A, I	5-10
Load Floating Point Multiple	72	LME	F, I	5-11
Load Floating Point Register	28	LER	C, A, I	5-10
Load Halfword	48	LH	C	2-6

APPENDIX 2 (Continued)

INSTRUCTION	OP-CODE	MNEMONIC	ATTRIBUTES	PAGE NO.
Load Halfword Immediate	C8	LHI	C	2-6
Load Halfword Logical	73	LHL	C	2-9
Load Immediate	F8	LI	C	2-5
Load Immediate Short	24	LIS	C	2-5
Load Multiple	D1	LM	F	2-10
Load Multiple Double Precision Floating Point	7F	LMD	D, I	5-26
Load Program Status Word	C2	LPSW	C, D, P, IA	6-12
Load Program Status Word Register	18	LPSWR	C, P, IA	6-13
Load Real Address	63	LRA	C, F, I	2-8
Load Register	08	LR	C	2-5
Load Register Double Precision Floating Point	38	LDR	C, A, I	5-25
Move and Process Byte String Register	30	MPBSR	C, I	9-5
Multiply	5C	M	F	4-12
Multiply Double Precision Floating Point	7C	MD	C, D, A, I	5-32
Multiply Floating Point	6C	ME	C, F, A, I	5-19
Multiply Floating Point Register	2C	MER	C, A, I	5-19
Multiply Halfword	4C	MH	H	4-13
Multiply Halfword Register	0C	MHR		4-13
Multiply Register	1C	MR		4-12
Multiply Register Double Precision Floating Point	3C	MDR	C, A, I	5-32
OR	56	O	C, F	2-22
OR Halfword	46	OH	C, H	2-24
OR Halfword Immediate	C6	OHI	C	2-24
OR Immediate	F6	OI	C	2-23
OR Register	06	OR	C	2-23
Output Command	DE	OC	C, P, IA	7-4
Output Command Register	9E	OCR	C, P, IA	7-4
Process Byte	62	PB	H, I	9-2
Process Byte Register	32	PBR	I	9-5
Read Block	D7	RB	C, F, P	7-9
Read Block Register	97	RBR	C, P	7-8
Read Data	DB	RD	C, P	7-6
Read Data Register	9B	RDR	C, P	7-6
Read Halfword	D9	RH	C, H, P	7-7
Read Halfword Register	99	RHR	C, P	7-7
Remove from Bottom of List	67	RBL	C, F, RP	2-46
Remove from Top of List	66	RTL	C, F, RP	2-46
Reset Bit	76	RBT	C, RP	2-39
Rotate Left Logical	EB	RLL	C	2-33
Rotate Right Logical	EA	RRL	C	2-34
Sense Status Arithmetic	DD	SS	C, P	7-5
Sense Status Register	9D	SSR	C, P	7-5
Set Bit	75	SBT	C, RP	2-37
Shift Left Arithmetic	EF	SLA	C	4-18
Shift Left Halfword Arithmetic	CF	SLHA	C	4-19

APPENDIX 2 (Continued)

INSTRUCTION	OP-CODE	MNEMONIC	ATTRIBUTES	PAGE NO.
Shift Left Halfword Logical	CD	SLHL	C	2-31
Shift Left Halfword Logical Short	91	SLHLS	C	2-31
Shift Left Logical	ED	SLL	C	2-29
Shift Left Logical Short	11	SLLS	C	2-29
Shift Right Arithmetic	EE	SRA	C	4-20
Shift Right Halfword Arithmetic	CE	SRHA	C	4-21
Shift Right Halfword Logical	CC	SRHL	C	2-32
Shift Right Halfword Logical Short	90	SRHLS	C	2-32
Shift Right Logical	EC	SRL	C	2-30
Shift Right Logical Short	10	SRLS	C	2-30
Simulate Channel Program	E3	SCP	C, F, P	7-15
Simulate Interrupt	E2	SINT	C, P, IA	6-15
Store	50	ST	F, RP	2-14
Store Byte	D2	STB	RP	2-17
Store Byte Register	92	STBR		2-17
Store Double Precision Floating Point	70	STD	D, RP, I	5-27
Store Floating Point	60	STE	F, RP, I	5-12
Store Floating Point Multiple	71	STME	F, RP, I	5-13
Store Halfword	40	STH	H, RP	2-15
Store Multiple	D0	STM	F, RP	2-35
Store Multiple Double Precision Floating Point	7E	STMD	D, RP, I	5-28
Subtract	5B	S	C, F	4-8
Subtract Double Precision Floating Point	7B	SD	C, D, A, I	5-30
Subtract Floating Point	6B	SE	C, F, A, I	5-16
Subtract Floating Point Register	2B	SER	C, A, I	5-16
Subtract Halfword	4B	SH	C, H	4-9
Subtract Halfword Immediate	CB	SHI	C	4-9
Subtract Immediate	FB	SI	C	4-8
Subtract Immediate Short	27	SIS	C	4-8
Subtract Register	0B	SR	C	4-8
Subtract Register Double Precision Floating Point	3B	SDR	C, A, I	5-30
Supervisor Call	E1	SVC	C, F	6-16
Test and Set	E0	TS	C, RP	2-35
Test Bit	74	TBT	C	2-36
Test Halfword Immediate	C3	THI	C	2-28
Test Immediate	F3	TI	C	2-27
Translate	E7	TLATE	F	2-42
Write Block	D6	WB	C, F, P	7-12
Write Block Register	96	WBR	C, P	7-13
Write Data	DA	WD	C, P	7-10
Write Data Register	9A	WDR	C, P	7-10
Write Halfword	D8	WH	C, H, P	7-11
Write Halfword Register	98	WHR	C, P	7-11

APPENDIX 3

INSTRUCTION SUMMARY - NUMERICAL

OP-CODE	MNEMONIC	INSTRUCTION	PAGE NO.
01*	BALR	Branch and Link Register	3-5
02*	BTCR	Branch on True Condition Register	3-3
03*	BFCR	Branch on False Condition Register	3-4
04	NR	AND Register	2-21
05	CLR	Compare Logical Register	2-18
06	OR	OR Register	2-23
07	XR	Exclusive OR Register	2-25
08	LR	Load Register	2-5
09	CR	Compare Register	4-10
0A	AR	Add Register	4-4
0B	SR	Subtract Register	4-8
0C*	MHR	Multiply Halfword Register	4-13
0D*	DHR	Divide Halfword Register	4-16
10	SRLS	Shift Right Logical Short	2-30
11	SLLS	Shift Left Logical Short	2-29
12	CHVR	Convert to Halfword Value	4-22
18	LPSWR	Load Program Status Word Register	6-13
1C*	MR	Multiply Register	4-12
1D*	DR	Divide Register	4-14
20*	BTBS	Branch on True Condition Backward Short	3-3
21*	BTFS	Branch on True Condition Forward Short	3-3
22*	BFBS	Branch on False Condition Backward Short	3-4
23*	BFFS	Branch on False Condition Forward Short	3-4
24	LIS	Load Immediate Short	2-5
25	LCS	Load Complement Short	2-5
26	AIS	Add Immediate Short	4-4
27	SIS	Subtract Immediate Short	4-8
28	LER	Load Floating Point	5-10
29	CER	Compare Floating Point	5-18
2A	AER	Add Floating Point Register	5-14
2B	SER	Subtract Floating Point Register	4-16
2C	MER	Multiply Floating Point Register	5-19
2D	DER	Divide Floating Point Register	5-21
2E	FXR	Fix Register	5-23
2F	FLR	Float Register	5-24
30	MPBSR	Move & Process Byte String Register	9-5
32*	PBR	Process Byte Register	9-5

* Condition Code Not Changed

APPENDIX 3 (Continued)

OP-CODE	MNEMONIC	INSTRUCTION	PAGE NO.
34*	EXHR	Exchange Halfword Register	2-13
38	LDR	Load Register Double Precision Floating Point	5-25
39	CDR	Compare Register Double Precision Floating Point	5-31
3A	ADR	Add Register Double Precision Floating Point	5-29
3B	SDR	Subtract REgister Double Precision Floating Point	5-30
3C	MDR	Multiply Register Double Precision Floating Point	5-32
3D	DDR	Divide Register Double Precision Floating Point	5-33
3E	FXDR	Fix Register Double Precision Floating Point	5-34
3F	FLDR	Float Register Double Precision Floating Point	5-35
40*	STH	Store Halfword	2-15
41*	BAL	Branch and Link	3-5
42*	BTC	Branch on True Condition	3-3
43*	BFC	Branch on False Condition	3-4
44	NH	AND Halfword	2-22
45	CLH	Compare Logical Halfword	2-19
46	OH	OR Halfword	2-24
47	XH	Exclusive OR Halfword	2-26
48	LH	Load Halfword	2-6
49	CH	Compare Halfword	4-11
4A	AH	Add Halfword	4-5
4B	SH	Subtract Halfword	4-9
4C*	MH	Multiply Halfword	4-13
4D*	DH	Divide Halfword	4-16
50*	ST	Store	2-14
51	AM	Add to Memory	4-6
54	N	AND	2-21
55	CL	Compare Logical	2-18
56	O	OR	2-22
57	X	Exclusive OR	2-25
58	L	Load	2-5
59	C	Compare	4-10
5A	A	Add	4-4
5B	S	Subtract	4-8
5C*	M	Multiply	4-12
5D*	D	Divide	4-14
5E*	CRC12	Cyclic Redundancy Check Modulo 12	2-40
5F*	CRC16	Cyclic Redundancy Check Modulo 16	2-40
60*	STE	Store Floating Point	5-12
61	AHM	Add Halfword to Memory	4-7
62*	PB	Process Byte	9-2
63	LRA	Load Read Address	2-8
64	ATL	Add to Top of List	2-45
65	ABL	Add to Bottom of List	2-45
66	RTL	Remove from Top of List	2-46
67	RBL	Remove from Bottom of List	2-46

* Condition Code Not Changed

OP-CODE	MNENONIC	INSTRUCTION	PAGE NO.
68	LE	Load Floating Point	5-10
69	CE	Compare Floating Point	5-18
6A	AE	Add Floating Point	5-14
6B	SE	Subtract Floating Point	5-16
6C	ME	Multiply Floating Point	5-19
6D	DE	Divide Floating Point	5-21
70*	STD	Store Double Precision Floating Point	5-27
71*	STME	Store Floating Point Multiple	5-13
72*	LME	Load Floating Point Multiple	5-11
73	LHL	Load Halfword Logical	2-9
74	TBT	Test Bit	2-36
75	SBT	Set Bit	2-37
76	RBT	Reset Bit	2-39
77	CBT	Complement Bit	2-38
78	LD	Load Double Precision Floating Point	5-25
79	CD	Compare Double Precision Floating Point	5-31
7A	AD	Add Double Precision Floating Point	5-29
7B	SD	Subtract Double Precision Floating Point	5-30
7C	MD	Multiply Double Precision Floating Point	5-32
7D	DD	Divide Double Precision Floating Point	5-33
7E*	STMD	Store Multiple Double Precision Floating Point	5-28
7F*	LMD	Load Multiple Double Precision Floating Point	5-26
90	SRHLS	Shift Right Halfword Logical Short	2-32
91	SLHLS	Shift Left Halfword Logical Short	2-31
92*	STBR	Store Byte Register	2-17
93*	LBR	Load Byte Register	2-11
94*	EXBR	Exchange Byte Register	2-12
95	EPSR	Exchange Program Status Word	6-14
96	WBR	Write Block Register	7-13
97	RBR	Read Block Register	7-9
98	WHR	Write Halfword Register	7-11
99	RHR	Read Halfword Register	7-7
9A	WDR	Write Data Register	7-10
9B	RDR	Read Data Register	7-6
9D	SSR	Sense Status Register	7-5
9E	OCR	Output Command Register	7-4
C0*	BXH	Branch on Index High	3-7
C1*	BXLE	Branch on Index Low or Equal	3-6
C2	LPSW	Load Program Status Word	6-12
C3	THI	Test Halfword Immediate	2-28
C4	NHI	AND Halfword Immediate	2-22
C5	CLHI	Compare Logical Halfword Immediate	2-19

* Condition Code Not Changed

OP-CODE	MNEMONIC	INSTRUCTION	PAGE NO.
C6	OHI	OR Halfword Immediate	2-24
C7	XHI	Exclusive OR Halfword Immediate	2-26
C8	LHI	Load Halfword Immediate	2-6
C9	CHI	Compare Halfword Immediate	4-11
CA	AHI	Add Halfword Immediate	4-5
CB	SHI	Subtract Halfword Immediate	4-9
CC	SRHL	Shift Right Halfword Logical	2-32
CD	SLHL	Shift Left Halfword Logical	2-31
CE	SRHA	Shift Right Halfword Arithmetic	4-21
CF	SLHA	Shift Left Halfword Arithmetic	4-19
D0*	STM	Store Multiple	2-35
D1*	LM	Load Multiple	2-10
D2*	STB	Store Byte	2-17
D3*	LB	Load Byte	2-11
D4	CLB	Compare Logical Byte	2-19
D5	AL	Autoload	7-14
D6	WB	Write Block	7-12
D7	RB	Read Block	7-8
D8	WH	Write Halfword	7-11
D9	RH	Read Halfword	7-7
DA	WD	Write Data	7-10
DB	RD	Read Data	7-6
DD	SS	Sense Status	7-5
DE	OC	Output Command	7-4
E0	TS	Test and Set	2-35
E1	SVC	Supervisor Call	6-16
E2	SINT	Simulate Interrupt	6-15
E3	SCP	Simulate Channel Program	7-15
E6*	LA	Load Address	2-7
E7*	TLATE	Translate	2-42
EA	RRL	Rotate Right Logical	2-34
EB	RLL	Rotate Left Logical	2-33
EC	SRL	Shift Right Logical	2-30
ED	SLL	Shift Left Logical	2-29

* Condition Code Not Changed

APPENDIX 3 (Continued)

OP-CODE	MNEMONIC	INSTRUCTION	PAGE NO.
EE	SRA	Shift Right Arithmetic	4-20
EF	SLA	Shift Left Arithmetic	4-18
F3	TI	Test Immediate	2-27
F4	NI	AND Immediate	2-21
F5	CLI	Compare Logical Immediate	2-18
F6	OI	OR Immediate	2-23
F7	XI	Exclusive OR Immediate	2-25
F8	LI	Load Immediate	2-5
F9	CI	Compare Immediate	4-10
FA	AI	Add Immediate	4-4
FB	SI	Subtract Immediate	4-8

APPENDIX 4
EXTENDED BRANCH MNEMONICS

INSTRUCTION	OP CODE (M1)	MNEMONIC	OPERAND
Branch on Carry	428	BC	A(X2)
Branch on Carry Register	028	BCR	R2
Branch on No Carry	438	BNC	A(X2)
Branch on No Carry Register	038	BNCR	R2
Branch on Equal	433	BE	A(X2)
Branch on Equal Register	033	BER	R2
Branch on Not Equal	423	BNE	A(X2)
Branch on Not Equal Register	023	BNER	R2
Branch on Low	428	BL	A(X2)
Branch on Low Register	028	BLR	R2
Branch on Not Low	438	BNL	A(X2)
Branch on Not Low Register	038	BNLR	R2
Branch on Minus	421	BM	A(X2)
Branch on Minus Register	021	BMR	R2
Branch on Not Minus	431	BNM	A(X2)
Branch on Not Minus Register	031	BNMR	R2
Branch on Plus	422	BP	A(X2)
Branch on Plus Register	022	BPR	R2
Branch on Not Plus	432	BNP	A(X2)
Branch on Not Plus Register	032	BNPR	R2
Branch on Overflow	424	BO	A(X2)
Branch on Overflow Register	024	BOR	R2
Branch on No Overflow	434	BNO	A(X2)
Branch on No Overflow Register	034	BNOR	R2
Branch Unconditional	430	B	A(X2)
Branch Unconditional Register	030	BR	R2
Branch on Zero	433	BZ	A(X2)
Branch on Zero Register	033	BZR	R2
Branch on Not Zero	423	BNZ	A(X2)
Branch on Not Zero Register	023	BNZR	R2
No Operation	420	NOP	
No Operation Register	020	NOPR	
Branch on Carry Short	208	BCS	A (Backward Reference)
	218	BCS	A (Forward Reference)
Branch on No Carry Short	228	BNCS	A (Backward Reference)
	238	BNCS	A (Forward Reference)
Branch on Equal Short	223	BES	A (Backward Reference)
	233	BES	A (Forward Reference)
Branch on Not Equal Short	203	BNES	A (Backward Reference)
	213	BNES	A (Forward Reference)
Branch on Low Short	208	BLS	A (Backward Reference)
	218	BLS	A (Forward Reference)
Branch on Not Low Short	228	BNLS	A (Backward Reference)
	238	BNLS	A (Forward Reference)

APPENDIX 4 (Continued)

INSTRUCTION	OP CODE (M1)	MNEMONIC	OPERANDS
Branch on Minus Short	201	BMS	A (Backward Reference)
	211	BMS	A (Forward Reference)
Branch on Not Minus Short	221	BNMS	A (Backward Reference)
	231	BNMS	A (Forward Reference)
Branch on Plus Short	202	BPS	A (Backward Reference)
	212	BPS	A (Forward Reference)
Branch on Not Plus Short	222	BNPS	A (Backward Reference)
	232	BNPS	A (Forward Reference)
Branch on Overflow Short	204	BOS	A (Backward Reference)
	214	BOS	A (Forward Reference)
Branch on No Overflow Short	224	BNOS	A (Backward Reference)
	234	BNOS	A (Forward Reference)
Branch Unconditional Short	220	BS	A (Backward Reference)
	230	BS	A (Forward Reference)
Branch on Zero Short	223	BZS	A (Backward Reference)
	233	BZS	A (Forward Reference)
Branch on Not Zero Short	203	BNZS	A (Backward Reference)
	213	BNZS	A (Forward Reference)

APPENDIX 5
ARITHMETIC REFERENCES

TABLE OF POWERS OF TWO

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5

APPENDIX 5 (Continued)

TABLE OF POWERS OF SIXTEEN

16^n							n
						1	0
						16	1
						256	2
					4	096	3
					65	536	4
				1	048	576	5
				16	777	216	6
				268	435	456	7
			4	294	967	296	8
			68	719	476	736	9
		1	099	511	627	776	10
		17	592	186	044	416	11
		281	474	976	710	656	12
	4	503	599	627	370	496	13
	72	057	594	037	927	936	14
1	152	921	504	606	846	976	15

Decimal Values

HEXADICIMAL TO DECIMAL INTEGER CONVERSION TABLE

BYTE				BYTE			
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0
1	4,096	1	256	1	16	1	1
2	8,192	2	512	2	32	2	2
3	12,288	3	768	3	48	3	3
4	16,384	4	1,024	4	64	4	4
5	20,480	5	1,280	5	80	5	5
6	24,576	6	1,536	6	96	6	6
7	28,672	7	1,792	7	112	7	7
8	32,768	8	2,048	8	128	8	8
9	36,864	9	2,304	9	144	9	9
A	40,960	A	2,560	A	160	A	10
B	45,056	B	2,816	B	176	B	11
C	49,152	C	3,072	C	192	C	12
D	53,248	D	3,328	D	208	D	13
E	57,344	E	3,584	E	224	E	14
F	61,440	F	3,840	F	240	F	15

APPENDIX 5 (Continued)

HEXADECIMAL ADDITION AND SUBTRACTION TABLE

Examples: 5+A = F; 18-D = B; A+B = 15

	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	1
2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	2
3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	3
4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	4
5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	5
6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	6
7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	7
8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	8
9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	9
A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	A
B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	B
C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	C
D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	D
E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	E
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	F
	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

HEXADECIMAL MULTIPLICATION AND DIVISION TABLE

Examples: 5x6 = 1E; 75÷D = 9; 58 ÷ 8 = B; 9xC = 6C

	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	1
2	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E	2
3	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D	3
4	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C	4
5	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B	5
6	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A	6
7	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69	7
8	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78	8
9	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87	9
A	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96	A
B	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5	B
C	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4	C
D	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3	D
E	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2	E
F	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1	F
	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

APPENDIX 5 (Continued)

TABLE OF MATHEMATICAL CONSTANTS

CONSTANT	DECIMAL VALUE				HEXADECIMAL VALUE		FLOATING POINT VALUE			
							DOUBLE PRECISION			
							SINGLE PRECISION			
π	3.14159	26535	89793	23846	3.243F	6A89	4132	43F6	A888	5A31
π-1	0.31830	98861	83790	67153	0.517C	C1B7	4051	7CC1	B727	220B
$\sqrt{\pi}$	1.77245	38509	05516	02729	1.C5BF	891C	411C	5BF8	91B4	EF6B
Ln π	1.14472	98858	49400	17414	1.250D	048F	4112	50D0	48E7	A1BD
$\sqrt{3}$	1.73205	08075	68877	29353	1.B67A	E858	411B	67AE	8584	CAA7
e	2.71828	18284	59045	23536	2.B7E1	5163	412B	7E15	1628	AED3
e^{-1}	0.36787	94411	71442	32159	0.5E2D	58D9	405E	2D58	D8B3	BCDF
$\sqrt{e}$	1.64872	12707	00128	14680	1.A612	98E2	411A	6129	8E1E	069C
$\log_{10}e$	0.43429	44819	03251	82765	0.6F2D	EC55	406F	2DEC	5A9B	9439
$\log_2 e$	1.44269	50408	88963		1.7154	7653	4117	1547	652	-----
γ	0.57721	56649	01532	86060	0.93C4	67E4	4093	C467	E37D	B0C8
Ln γ	-0.54953	93129	81644	82233	-0.8CAE	9BC1	C08C	AE9B	C11F	5A60
$\sqrt{2}$	1.41421	35623	73095	04880	1.6A09	E668	4116	A09E	667F	3BCD
Ln2	0.69314	71805	59945	30941	0.B172	17F8	40B1	7217	F7D1	CF7A
$\log_{10}2$	0.30102	99956	63981	19521	0.4D10	4D42	404D	104D	427D	E7FC
$\sqrt{10}$	3.16227	76601	68379	33200	3.298B	075C	4132	98B0	75B4	B6A5
Ln10	2.30258	50929	04945	68401	2.4D76	3777	4124	D763	776A	AA2B

INTEGER CONVERSION TABLE

Hexadecimal and Decimal Integer Conversion Table

HALFWORD								HALFWORD							
BYTE				BYTE				BYTE				BYTE			
BITS: 0123		4567		0123		4567		0123		4567		0123		4567	
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	1,280	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	167,772,160	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
8		7		6		5		4		3		2		1	

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.

EXAMPLE	
Conversion of Hexadecimal Value	D34
1. D	3328
2. 3	48
3. 4	4
4. Decimal	3380

To convert integer numbers greater than the capacity of table, use the techniques below:

HEXADECIMAL TO DECIMAL

Successive cumulative multiplication from left to right, adding units position.

Example: $D34_{16} = 3380_{10}$

$$\begin{array}{rr} D = & 13 \\ & \underline{x16} \\ & 208 \\ 3 = & \underline{+3} \\ & 211 \\ & \underline{x16} \\ & 3376 \\ 4 = & \underline{+4} \\ & 3380 \end{array}$$

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.

EXAMPLE	
Conversion of Decimal Value	3380
1. D	-3328 52
2. 3	-48 4
3. 4	-4
4. Hexadecimal	D34

DECIMAL TO HEXADECIMAL

Divide and collect the remainder in reverse order.

Example: $3380_{10} = X_{16}$

$$\begin{array}{lll} 16 \,\underline{|3380} & \searrow & \text{remainder} \\ 16 \,\underline{|211} & \searrow & 4 \uparrow \\ 16 \,\underline{|13} & \searrow & 3 \\ & & D \end{array}$$

$3380_{10} = D34_{16}$

APPENDIX 5 (Continued)
FRACTION CONVERSION TABLE

Hexadecimal and Decimal Fraction Conversion Table

HALFWORD													
BYTE					BYTE								
BITS	0123	4567			0123				4567				
Hex	Decimal	Hex	Decimal		Hex	Decimal			Hex	Decimal Equivalent			
.0	.0000	.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 7500

.ABC Hex is equal to .6708 9843 7500

TO CONVERT .13 DECIMAL TO HEXADECIMAL

1. Find .1250 next lowest to subtract
   .1300
   -.1250 = .2 Hex
2. Find .0039 0625 next lowest to
   .0050 0000
   -.0039 0625 = .01
3. Find .0009 7656 2500
   .0010 9375 0000
   -.0009 7656 2500 = .004
4. Find .0001 0681 1523 4375
   .0001 1718 7500 0000
   -.0001 0681 1523 4375 = .0007
   .0000 1037 5976 5625 = .2147 Hex
5. 13 Decimal is approximately equal to ——————↑

To convert fractions beyond the capacity of table, use techniques below:

HEXADECIMAL FRACTION TO DECIMAL

Convert the hexadecimal fraction to its decimal equivalent using the same technique as for integer numbers. Divide the results by 16^n (n is the number of fraction positions).

Example: $.8A7 = .540771_{10}$

$8A7_{16} = 2215_{10}$

$16^3 = 4096$

$$\begin{array}{r} .540771 \\ 4096\overline{)2215.000000} \end{array}$$

DECIMAL FRACTION TO HEXADECIMAL

Collect integer parts of product in the order of calculation.

Example: $.5408_{10} = .8A7_{16}$

$$\begin{array}{lr} & .5408 \\ & \underline{\times 16} \\ 8 \leftarrow & \boxed{8}.6528 \\ & \underline{\times 16} \\ A \leftarrow & \boxed{10}.4448 \\ & \underline{\times 16} \\ \downarrow 7 \leftarrow & \boxed{7}.1168 \end{array}$$

APPENDIX 6

MODEL 7/32 EXECUTION TIMES IN MICRO SECONDS (1000 NANOSECOND MEMORY)

INST.	EXECUTION TIME	NOTES	COMMENTS
A	3.50	1	
ABL	5.00/9.50/9.75	1	OVF/NORM/WRAP
AD			
ADR			
AE	12.75/17.75/23.25	1	MIN/AVG/MAX
AER	11.75/16.75/22.25	1	MIN/AVG/MAX
AH	2.75	1	
AHI	1.75		
AHM	3.75	1	
AI	2.75		
AIS	1.25		
AL	6.25+2.50L+2.50n	5	L = LEADER, n = BYTES
AM	5.75	1	
AR	1.00		
ATL	5.00/9.75/9.75	1	OVF/NORM/WRAP
BAL	2.00	1	
BALR	1.50		
BFBS	1.50/2.00		No branch/branch
BFC	2.00/2.00	4	No branch/branch
BFCR	1.50/1.50		No branch/branch
BFFS	1.50/2.00		No branch/branch
BTBS	1.50/2.00		No branch/branch
BTC	2.00/2.00	4	No branch/branch
BTCR	1.50/1.50		No branch/branch
BTFS	1.50/2.00		No branch/branch
BXH	4.75/4.25	1	No branch/branch
BXLE	4.75/4.25	3	No branch/branch

APPENDIX 6
(CONTINUED)

INST.	EXECUTION TIME	NOTES	COMMENTS
C	4.50/4.25		SIGNS ALIKE/DIFFER
CBT	6.25/7.00/7.75	1	MIN/AVG/MAX
CD			
CDR			
CE	7.25/8.25/9.50	1	MIN/AVG/MAX
CER	6.25/7.25/8.50		MIN/AVG/MAX
CH	3.75/3.50		SIGNS ALIKE/DIFFER
CHI	2.75/2.50		SIGNS ALIKE/DIFFER
CHVR	2.75/4.00		NORM/OVF
CI	3.75/3.50		SIGNS ALIKE/DIFFER
CL	3.50	1	
CLB	3.00	1	
CLH	2.75	1	
CLHI	1.75		
CLI	2.75		
CLR	1.00		
CR	2.00/1.75		SIGNS ALIKE/DIFFER
CRC12	11.50/13.25/15.00	1	MIN/AVG/MAX
CRC16	13.00/15.25/17.50	1	MIN/AVG/MAX
D	82.75/88.50/96.50	1	MIN/AVG/MAX
DD			
DDR			
DE	49.25/50.25/51.00	1	MIN/AVG/MAX
DER	48.25/48.25/50.00		MIN/AVG/MAX
DH	13.00/13.00/14.25	1	MIN/AVG/MAX
DHR	11.00/11.00/12.25		MIN/AVG/MAX
DR	80.25/86.00/94.00		MIN/AVG/MAX
EPSR	3.75		
EXBR	1.00		
EXHR	1.00		

APPENDIX 6
(CONTINUED)

INST.	EXECUTION TIME	NOTE	COMMENTS
FLDR			
FLR	10.50/13.75/18.00		MIN/AVE/MAX
FXDR			
FXR	8.25/10.25/16.25		MIN/AVG/MAX
L	3.50	1	
LA	2.25	5	
LB	3.00	1	
LBR	1.25		
LCS	1.50		
LD			
LDR			
LE	7.50/10.50/14.25	1	MIN/AVG/MAX
LER	6.50/9.50/13.25		MIN/AVG/MAX
LH	2.75	1	
LHI	1.75		
LHL	2.75	1	
LI	2.75		
LIS	1.00		
LM	2.75+2.25n	1	n = REGISTERS
LMD			
LME	7.25+3.25 (n-1)	1,2	n = REGISTERS
LPSW	7.25	1	
LPSWR	3.50		
LR	1.00		
LRA	5.50/8.25/8.25/8.50/8.25	5	LIMIT/PRESENCE/WRITE/EXEC/ NORMAL
M	24.25/25.25/26.75	1	MIN/AVG/MAX
MD			
MDR			
ME	30.00/30.25/32.00	1	MIN/AVG/MAX
MER	29.00/29.25/31.00	1	MIN/AVG/MAX
MH	6.25	1	
MHR	4.25		
MPBSR	See A6-8		
MR	21.75/22.75/24.25		MIN/AVG/MAX
N	3.50	1	
NH	2.75	1	

APPENDIX 6
(CONTINUED)

INST.	EXECUTION TIME	NOTES	COMMENTS
NHI	1.75		
NI	2.75		
NR	1.00		
O	3.50	1	
OC	4.00	1	
OCR	3.25		
OH	2.75	1	
OHI	1.75		
OI	2.75		
OR	1.00		
PB	4.75	1	
PBR	3.50		
RB	7.25+2.50n	1	n = BYTES
RBL	5.00/9.75/9.75	1	OVF/NORM/WRAP
RBR	4.50+2.50n		n = BYTES
RBT	6.75/7.00/7.75	1	MIN/AVG/MAX
RD	4.25	1	
RDR	2.25		
RH	4.75/4.00	1	BYTE/HALFWORD
RHR	3.00/2.25		BYTE/HALFWORD
RLL	2.25/1.75+1.00n		n=0/n = SHIFTS
RRL	2.25/1.75+1.00n		n=0/n = SHIFTS
RTL	5.00/10.75/11.00	1	OVF/NORM/WRAP
S	3.50	1	
SBT	6.75/7.00/7.75	1	MIN/AVG/MAX
SCP	5.75/9.50/9.25/11.00	1	CNT+/NORM/TERM/TERM (FAST) (NORM)
SD			
SDR			
SE	13.25/18.25/23.75	1	MIN/AVG/MAX
SER	12.25/17.25/22.75		MIN/AVG/MAX

APPENDIX 6
(CONTINUED)

INST.	EXECUTION TIME	NOTES	COMMENTS
SH	2.75	1	
SHI	1.75		
SI	2.75		
SINT	7.75		IMMEDIATE INTERRUPT
SIS	1.25		
SLA	3.75+ (n-2)/2 (.25)	3	n = SHIFTS
SLHA	2.75+(n-1) (.25)	2	n = SHIFTS
SLHL	2.50+(n-1) (.25)	2	n = SHIFTS
SLHLS	1.75+(n-1) (.25)	2	n = SHIFTS
SLL	3.25+ (n-2)/2 (.25)	3	n = SHIFTS
SLLS	2.75+ (n-2)/2 (.25)	3	n = SHIFTS
SR	1.00		
SRA	3.75+ (n-2)/2 (.25)	3	n = SHIFTS
SRHA	2.50+(n-1) (.25)	2	n = SHIFTS
SRHL	2.50+(n-1) (.25)	2	n = SHIFTS
SRHLS	1.75+(n-1) (.25)	2	n = SHIFTS
SRL	3.25+ (n-2)/2 (.25)	3	n = SHIFTS
SRLS	2.75+ (n-2)/2 (.25)	3	n = SHIFTS
SS	4.50	1	
SSR	3.00		
ST	3.75	1	
STB	3.75	1	
STBR	2.00		
STD STE	5.75	1	
STH	2.75	1	
STM	2.50+2.00n	1	n = REGISTERS
STMD STME	7.50+3.50(n-1)	1,2	n = REGISTERS
SVC	7.00	1	

APPENDIX 6
(CONTINUED)

INST.	EXECUTION TIME	NOTES	COMMENTS
TBT	6.00/6.75/7.50	1	MIN/AVG/MAX
THI	1.75		
TI	2.75		
TLATE	4.25/5.25	1	Translation/Spec. Character Bit Set/Bit Reset
TS	3.50/4.00	1	
WB	7.25/2.75n	1	
WBR	4.50/2.75n		
WD	3.75	1	
WDR	2.25		
WH	4.75/3.75	1	BYTE/HALFWORD
WHR	3.75/2.50		BYTE/HALFWORD
X	3.50	1	
XH	2.75	1	
XHI	1.75		
XI	2.75		
XR	1.00		

All execution times assume no DMA interference. Times given for I/O Instructions assume best case device response.

NOTES

1. Add 1.00 if RX3 format
2. (n-1) is zero if n is zero
3. (n-2)/2 is zero if n is 0, 1, 2 or 3.
4. If branch is taken, add 0.75 if RX3.
5. Add 0.75 if RX3 format

APPENDIX 6
(CONTINUED)

AUTO DRIVER CHANNEL EXECUTION TIMES IN MICROSECONDS

FAST MODE

FUNCTION	EXECUTE RESET	BAD STATUS	BYTE COUNT PLUS	NORMAL	BUFFER END
READ (BYTE)	9.50	10.75	12.25	16.25	16.50
READ (HALFWORD)	9.50	10.75	12.25	16.75	17.00
WRITE (BYTE)	9.50	10.75	12.25	16.00	16.25
WRITE (HALFWORD)	9.50	10.75	12.25	17.00	17.25

NORMAL MODE

FUNCTION	EXECUTE RESET	BAD STATUS	BYTE COUNT PLUS	NORMAL	BUFFER END	NOTE
LRC, BUFF0, READ	9.50	10.75	13.25	21.25	21.00	
LRC, BUFF0, READ, TLATE	9.50	10.75	13.25	25.25/20.50	16.00	1
LRC, BUFF0, WRITE	9.50	10.75	13.25	20.75	21.50	
LRC, BUFF0, WRITE, TLATE	9.50	10.75	13.25	24.50/20.00	25.25	1
LRC, BUFF1, READ	9.50	10.75	14.00	21.00	22.75	
LRC, BUFF1, READ, TLATE	9.50	10.75	14.00	26.00/21.25	26.75	1
LRC, BUFF1, WRITE	9.50	10.75	14.00	21.50	22.25	
LRC, BUFF1, WRITE, TLATE	9.50	10.75	14.00	25.25/21.50	26.00	1
CRC, BUFF0, READ	9.50	10.75	13.25	24.75	25.50	2
CRC, BUFF0, READ, TLATE	9.50	10.75	13.25	28.75/20.50	29.50	1,2
CRC, BUFF0, WRITE	9.50	10.75	13.25	24.25	25.00	2
CRC, BUFF0, WRITE, TLATE	9.50	10.75	13.25	28.00/20.75	28.75	1,2
CRC, BUFF1, READ,	9.50	10.75	14.00	25.50	26.25	2
CRC, BUFF1, READ, TLATE	9.50	10.75	14.00	29.50/21.25	30.25	1,2
CRC, BUFF1, WRITE	9.50	10.75	14.00	25.00	25.75	2
CRC, BUFF1, WRITE, TLATE	9.50	10.75	14.00	28.75/21.50	29.50	1,2

NOTE 1 NORMAL/SPECIAL CHARACTER

NOTE 2 If data communication option is not equipped, add to the NORMAL time 5.00/7.25/9.50 MIN/AVG/MAX

IMMEDIATE INTERRUPTS	5.75
MALF	15.00
MAC	8.00
ILLEGAL INSTR	7.50
on LPSW, LPSWR, EPSR ADD	3.50 IF QUEUE SERVICE ENABLED
THEN ADD	7.75 IF QUEUE NOT EMPTY

MPBSR Instruction Execution Times

SEQUENCE	FIRST BYTE	SUBSEQUENT BYTES	SPECIAL CHARACTER	INTERRUPTED BYTE	LAST BYTE
ERROR CHECK ONLY	8.50	4.50		2.75	6.25
TRANSLATE ONLY	8.25	5.00	6.00	1.50	6.00
CHECK THEN TRANSLATE	10.00	5.50	7.25	2.50	7.25
TRANSLATE THEN CHECK	9.75	5.50	7.25	2.50	6.75

APPENDIX 7

I/O REFERENCES

TELETYPE ASCII/HEX CONVERSION TABLE

HEX (MSD) →					0	1	2	3	4	5	6	7
(LSD) ↓	Teletype Tape Channels → ↓			8	DEPENDS UPON PARITY*							
				7	0	0	0	0	1	1	1	1
				6	0	0	1	1	0	0	1	1
				5	0	1	0	1	0	1	0	1
	4	3	2	1								
0	0	0	0	0	NULL	DC_0	SPACE	0	@	P		
1	0	0	0	1	SOM	X-ON	!	1	A	Q		
2	0	0	1	0	EOA	TAPE ON	''	2	B	R		
3	0	0	1	1	EOM	X-OFF	#	3	C	S		
4	0	1	0	0	EOT	TAPE OFF	$	4	D	T		
5	0	1	0	1	WRU	ERR	%	5	E	U		
6	0	1	1	0	RU	SYNC	&	6	F	V		
7	0	1	1	1	BELL	LEM	'	7	G	W		
8	1	0	0	0	FE_0	S_0	(	8	H	X		
9	1	0	0	1	HT/SK	S_1	)	9	I	Y		
A	1	0	1	0	LF	S_2	*	:	J	Z		
B	1	0	1	1	VT	S_3	+	;	K	[		
C	1	1	0	0	FF	S_4	,	<	L	\		ACK
D	1	1	0	1	CR	S_5	-	=	M	]		ALT. MODE
E	1	1	1	0	SO	S_6	.	>	N	↑		ESC
F	1	1	1	1	SI	S_7	/	?	O	←		DEL

*Parity bit adjusted for even parity (even number of 1's) on input from Teletype keyboard. Parity bit is ignored on output to Teletype printer.

ASCII CARD CODE CONVERSION TABLE

GRAPHIC	7-BIT ASCII CODE	CARD CODE	GRAPHIC	7-BIT ASCII CODE	CARD CODE
SPACE	20	BLANK	@	40	8-4
!	21	11-8-2	A	41	12-1
"	22	8-7	B	42	12-2
#	23	8-3	C	43	12-3
$	24	11-8-3	D	44	12-4
%	25	0-8-4	E	45	12-5
&	26	12	F	46	12-6
'	27	8-5	G	47	12-7
(	28	12-8-5	H	48	12-8
)	29	11-8-5	I	49	12-9
*	2A	11-8-4	J	4A	11-1
+	2B	12-8-6	K	4B	11-2
,	2C	0-8-3	L	4C	11-3
-	2D	11	M	4D	11-4
.	2E	12-8-3	N	4E	11-5
/	2F	0-1	O	4F	11-6
0	30	0	P	50	11-7
1	31	1	Q	51	11-8
2	32	2	R	52	11-9
3	33	3	S	53	0-2
4	34	4	T	54	0-3
5	35	5	U	55	0-4
6	36	6	V	56	0-5
7	37	7	W	57	0-6
8	38	8	X	58	0-7
9	39	9	Y	59	0-8
:	3A	8-2	Z	5A	0-9
;	3B	11-8-6	[	5B	12-8-2
<	3C	12-8-4	\	5C	0-8-2
=	3D	8-6	]	5D	12-8-7
>	3E	0-8-6	↑	5E	11-8-7
?	3F	0-8-7	←	5F	0-8-5

STANDARD-PREFERRED ADDRESS TABLE

MSD ↓ / LSD →	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		DISPLAY	TTY-CAROUSEL 15, 30 CRT ON CLI	PAPER TAPE RDR ONLY	CARD READER	LOADER STORAGE UNIT									201/301 DATA SET HDX	201/301 DATA SET FDX
1	PALM PASLA			PUNCH ONLY READER/ PUNCH CCM.												
2	8 LINE INTERRUPT MODULE (ADRS 20 to 27)								SECOND 8 LINE INTERRUPT MODULE (ADRS 28 TO 2F)							
3		CONTACT CLOSURE MODULE	I/O BUS SWITCH												360/370 AUX. INF	360/370 INF
4						FIRST CASSETTE SYSTEM						DIGITAL MUX				
5																
6			LINE PRINTERS			SECOND CASSETTE SYSTEM							UNIVERSAL CLOCK VARIABLE 60Hz			
7		RELAY DRIVER MODULE														801 DIALER
8				CONVERSION EQUIPMENT		556/800 BPI MAG TAPE			AIC			ULI				
9									AOC							
A									DIO						MEMORY PROTECT CONTROL	
B							REMOVABLE CARTRIDGE DISC CONT		QSA							
C						1600 BPI MAG TAPE	DISC 0	FIXED DISC 0								
D							DISC 1	FIXED DISC 1								
E							DISC 2	FIXED DISC 2								
F	SELECTOR CHANNEL						DISC 3	FIXED DISC 3				40 MBYTE DICS SYSTEM	FILE 0	FILE 1	FILE 2	FILE 3

AIC = ANALOG INPUT CONTROLLER
AOC = ANALOG OUTPUT CONTROLLER
DIO = DIGITAL I/O CONTROLLER

QSA = QUAD SYNCHRONOUS ADAPTER
ULI = UNIVERSAL LOGIC INTERFACE

APPENDIX 7 (Continued)

CAROUSEL ASCII/HEX CONVERSION TABLE

BITS				b6 b5 b4	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b3 ↓	b2 ↓	b1 ↓	b0 ↓	MSD LSD	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SPACE	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	A	LF	SUB	*	:	J	Z	j	z
1	0	1	1	B	VT	ESC	+	;	K	[	k	{
1	1	0	0	C	FF	FS	,	<	L	\	l	¦
1	1	0	1	D	CR	GS	–	=	M	]	m	}
1	1	1	0	E	SO	RS	.	>	N	^	n	~
1	1	1	1	F	SI	US	/	?	O	—	o	DEL

NUL	Null	DLE	Data link escape
SOH	Start of heading	DC1-3	Device control
STX	Start of text	DC4	Device stop
ETX	End of text	NAK	Negative acknowledge
EOT	End of transmission	SYN	Synchronous idle
ENQ	Enquiry	ETB	End of transmission block
ACK	Acknowledge	CAN	Cancel
BEL	Audible signal	EM	End of medium
BS	Backspace	SUB	Start of special sequence
HT	Horizontal tabulation	ESC	Escape
LF	Line feed	FS	File separator
VT	Vertical tabulation	GS	Group separator
FF	Form feed	RS	Record separator
CR	Carrier return	US	Unit separator
SO	Shift out	SP	Space
SI	Shift in	DEL	Delete/Idle

INDEX

INDEX (Continued)

INDEX (Continued)

CUT ALONG LINE

PUBLICATION COMMENT FORM

Please use this postage-paid form to make any comments, suggestions, criticisms, etc. concerning this publication.

From ______________________ Date ______________________

Title ______________________ Publication Title ______________________

Company______________________ Publication Number ______________________

Address ______________________

____

____

FOLD FOLD

Check the appropriate item.

□ Error Page No. ________ Drawing No. ______________

□ Addition Page No. ________ Drawing No. ______________

□ Other Page No. ________ Drawing No. ______________

Explanation:

FOLD FOLD

Fold and Staple
No postage necessary if mailed in U.S.A.

STAPLE STAPLE

FOLD FOLD

FIRST CLASS
PERMIT No. 22
OCEANPORT, N. J.

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY:

INTERDATA®

Subsidiary of PERKIN-ELMER
Oceanport, New Jersey 07757, U.S.A.

TECH PUBLICATIONS DEPT. MS 322

FOLD FOLD

STAPLE STAPLE