DRAFT

11/60 MICROPROGRAMMING

SPECIFICATION

(DRAFT D, 11-1-77)

(Please direct comments to Tom Sherman
ML3-3/E71, x5300)

PREFACE

This manual is directed to the experienced assemby-
language programmer and to the hardware engineer with
some programming experience.

Although the approach is tutorial, and some introductory
information is included, this manual is not intended to
teach a higher-level language programmer how to micro-
program. A familiarity with the PDP-11, and with machine
organization in general, is assumed.

This manual describes the 11/60 as seen from the micro-
programming level. The cache, memory management, bus
control, and floating point hardware are not described
in detail.

A subset of the ISP notation is used in this manual to
describe hardware functions. This notation is described
in Appendix B. In programming examples, this ISP
notation is used as if it were source code. Note that
none of these examples will run on MICRO-11 or any other
microassembler without the proper field and macro
definitions.

Appendix C contains a selective annotated bibliography
of recent work on microprogramming.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The 11/60 is a user-microprogrammable PDP-11 central processor.
The Writable Control Store option, along with its associated
software tools, provides a means by which you can tailor the
CPU to your specific needs. The subject of this manual is the
hardware environment visible to the microprogrammer.

To provide a secure basis for understanding the detailed
information in later chapters, this chapter focuses on three
topics:

    1.   What is microprogramming?

    2.   What is a datapath?

    3.   User microprogramming on the 11/60.

A short review of terms and concepts of hardware, architecture,
and microprogramming, addressing the first two topics, precedes
the discussion of 11/60 microprogramming. The final section of
this chapter discusses the structure and scope of this manual.

1.1 WHAT IS MICROPROGRAMMING?

Microprogramming is a method of controlling the functions of a
computer. The essential ideas of microprogramming were first
outlined by M.V. Wilkes in 1951[1]. Wilkes proposed a structured
hardware design technique to replace prevailing ad hoc methods

---

[1] Wilkes, M.V., "The Best Way to Design an Automatic Calculating
Machine." Manchester Univ. Inaugural Conference, 1951, pp16-21.

of logic design. He observed that a machine-language instruction could be subdivided into a sequence of elementary operations which he called micro-operations. He likened the execution of the individual steps to the execution of the individual instructions in a program. This concept is the basis of all microprogramming.

For many years, microprogramming remained the province of the hardware designer. As new machines, incorporating advances in theory and technology, were designed, software for older, slower machines became obsolete. Microprogramming proved to be an attractive solution to this problem of incompatibility. New machines could be provided with additional read-only memory, or control store, which allowed them to emulate earlier computers. The use of emulation, or the interpretive execution of a foreign instruction set, was later extended to provide upward and downward compatibility among a number of computers in a family.

The IBM System 360 series was a landmark application of microprogramming to achieve compatibility. In this series, there is a common architecture, the 360, which is the target machine. The different models are 360 emulators implemented on different host machines. The performance range of the series is due to the varying characteristics of the different host machines.

Microprogramming as a tool of the user has evolved slowly. Three things had to happen before it became truly feasible. First, technological advances in the field of fast random-access memories was required. The use of read-only memories in a user environment was troublesome and expensive, because correction of programming errors, or bugs, required new memories. Second, user micro-

programming required the spread of previously specialized knowledge. When only those engineers actually involved in the design of microprogrammed computers knew what microprogramming involved, users and educators were at a severe disadvantage. In recent years, microprogramming has found a place in computer science curricula, and has been widely used throughout the electronics industry. The third, and most important prerequisite for user microprogramming is the inclusion of generality and extendability in the design of a computer. A machine designed solely to implement a given instruction set, and with no address space for user control programs, makes alteration an onerous task. A corollary to this point is that software tools must be developed, so that the user does not have to work solely with binary patterns.

1.1.1 THE DATAPATH OF A COMPUTER

The heart of the 11/60 is a three-board microprocessor, whose operational unit is the datapath. A datapath is composed of three types of components:

1. Combinational units, such as adders, decoders, or other logical circuits;

2. Sequential units, such as registers and counters;

3. Connections, such as wires.

The execution of a PDP-11 instruction involves a sequence of transfers from one register in the datapath to another; some of these transfers take place directly, others involve an adder or other logical circuit. Each step in this sequence is controlled by a microinstruction; a set of such microinstructions is known as a microprogram.

Microprograms are held in a control store, a block of high-speed memory which can be accessed once per machine cycle. (A machine cycle is the basic unit of time within a processor.)

The control of the hardware components of the datapath by a micro-program is best explained by a simple example.

1.1.2 A SIMPLE DATAPATH

Figure 1-1 shows a simplified datapath. Its only combinational component is an Arithmetic/Logic Unit (ALU), which has two inputs. The ALU result, or output, is stored in D, which is a temporary holding register. The other components of this data-path are B, another holding register; a scratchpad (SPAD), which is a collection of 16 holding registers; and their interconnections. The circles in the diagram indicate gating logic.
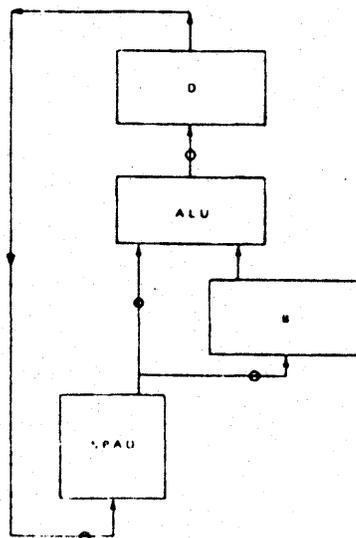
Figure 1-1 A Simplified Datapath

The arrows in the figure represent the flow of data within this datapath. Two operands are presented to the ALU inputs; the ALU combines these and presents the result at the input to D. After storage in D, the result can be presented at the input of one of the registers in the scratchpad.

To route the flow of data between the components of this data-path, a set of gates, with corresponding control signals, is required. The set of control signals needed is determined by the topology of the interconnections between the sequential and combinational units of the datapath.

For this datapath, the following control signals are needed.

     LOAD D - To store the ALU result in D

     ALUF   - To select the ALU function

     LOAD B - To store data from the scratchpad in B

     R/W   - To specify reading from or writing to the
              scratchpad registers

     ADDRS - To specify a location in the scratchpad

These signals are shown in Figure 1-2.

1.1.3 CONTROLLING THE DATAPATH

Now we can construct a microprogram to control this datapath. To perform a PDP-11 instruction, we must set up an initial constraint: the eight PDP-11 general registers will be stored in the first eight locations of the scratchpad. To perform the PDP-11 operation

     ADD R2, R1

the second and third locations in the scratchpad must be added,

and the result stored in the second location R [1]. Symbolically, this is represented as:

R[1] ← R[2] + R[1]

(The back-arrow symbol is read as "gets".)
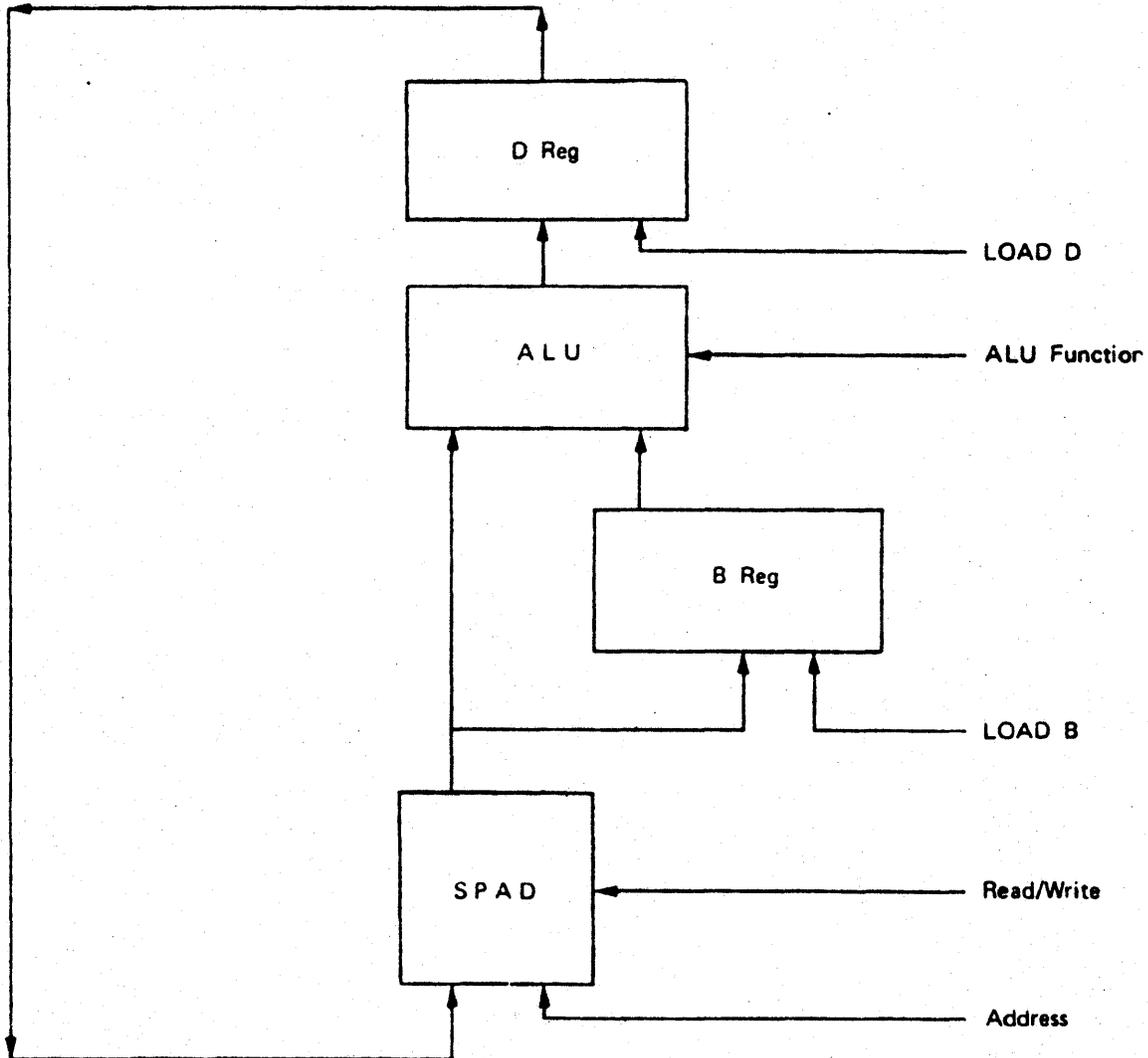
Figure 1-2 Simplified Datapath With Control Signals

It takes three steps, or machine cycles, to perform this operation with this simple datapath. This avoids conflicting data signals which would produce invalid results. First, R[2] is loaded into B; next, D is loaded with the sum of B and R[1]; and lastly, the result is written back to R[1]. The following are the basic machine steps:

CYCLE 1:   B ← R[2]

CYCLE 2:   D ← R[1]   PLUS B

CYCLE 3:   R[1] ← D

A time state table can be constructed to indicate which control signals must be asserted in each of these steps, as shown in Figure 1-3. The N/A entries indicate that the assertion of the signal will not afffect the current operation.

TIME	CONTROL SIGNALS				
	R/W	LOAD D	LOAD B	ALUF	ADDRESS
CYCLE 1	R	N/A	YES	N/A	R[2]
CYCLE 2	R	YES	NO	PLUS	R[1]
CYCLE 3	W	NO	N/A	N/A	R[1]

Bits: 2   1   1   4   4   =   12 total

Figure 1-3 Time State Table

After creating the time state table, we find that twelve bits are needed to provide the control signals for this datapath. The ALU

1-7

is allocated four bits to allow for a variety of operations; the scratchpad is assumed to have 16 locations, and the READ/WRITE signal is allocated two bits for a "do nothing" state.

These twelve bits can be combined to form a format for a micro-instruction.

11	10	9	8	7	6	5	4	3	2	1	0

- ADDRS
- ALUF
- LOAD B
- LOAD D
- WRITE
- READ

} These are the fields
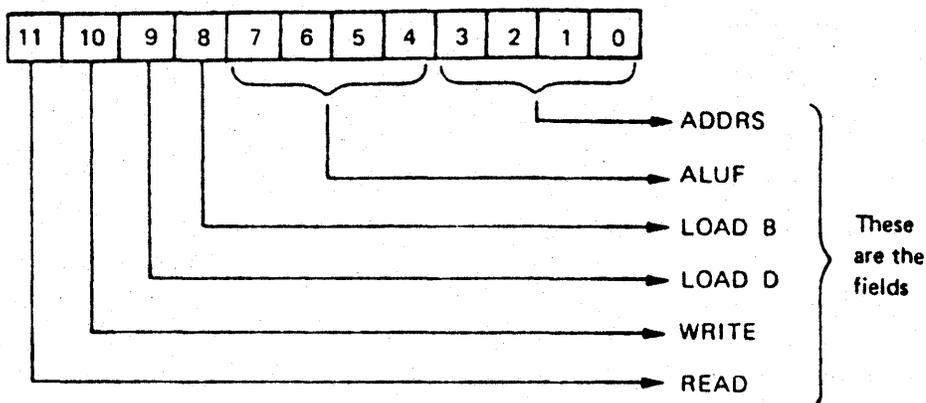
This microinstruction format, or microword, is divided into fields. Each field comprises the bits which are used to control a particular signal or function.

Using the time state table and the microinstruction format, we can now write a microprogram to perform the PDP-11 instruction ADD R2, R1:

CYCLE 1:	1	0	0	1	0	0	0	0	0	0	1	0
CYCLE 2:	1	0	1	0	0	0	0	1	0	0	0	1
CYCLE 3:	0	1	0	0	0	0	0	0	0	0	0	1

1.1.4 MICROPROGRAMMING AND MACHINE STATE

The general registers form part of the processor state of a PDP-11. By defining the first eight locations of the scratchpad as the PDP-11 general registers, we have made our simple datapath

implement, in part, a PDP-11.

The <u>processor state</u> of a computer is the set of registers and flags that hold the information left upon the completion of one instruction available for use during the execution of the next instruction.

Programmers working at different levels of a machine see different machine states: an applications programmer may never be concerned with machine state at all. A machine-language, or macro-level programmer knows the PDP-11 processor state to be defined by the contents of R∅ through R7 and the Processor Status Word. Nearly 100 registers are included in the machine state known to 11/60 microprogrammers. At the nano- or hardware level, even more machine state is seen.

This concept of machine, or processor, state is fundamental to an understanding of microprogrammable processors like the 11/60. State changes at the microprogramming level can affect the macro-level processor state.

For those readers with some exposure to the theory of finite-state machines, the analogy with a microprogrammed machine may be useful. A computer is made unique, or defined, by the functions it performs and the machine states it enters while performing those functions. Because of this, two machines can be built differently and yet perform identically. A microprogrammed machine changes state as it reads successive locations in the control store, emulating the state changes that would take place in a completely "hard-wired" machine. Additionally, the macro-level state, which

is a subset of the micro-level machine state, changes as if there were no machine but the macro-level machine. A PDP-11 is thus "covered" by an 11/60.

1.1.5 ARCHITECTURE AND ORGANIZATION

To additionally distinguish the macro-level machine from the micro-level machine, it is useful to differentiate between the terms architecture and organization.

Architecture, in this manual, refers to that set of a computer's features that are visible to the programmer. To a PDP-11 machine-language programmer, this includes the general registers, the instruction set, and the Processor Status Word. It was architectural identity that made the members of the IBM System 360 series compatible.

Organization describes a level below architecture, and is concerned with many items that are invisible to the programmer.

The term architecture describes what facilities are provided, while organization is concerned with how those facilities are provided. (Occasionally, another term is included in this hierarchy: realization. This term is used to characterize the components used in a particular machine implementation, such as the type of logic and chips used.)

The macro-level organization, transparent to the macro-level programmer, defines the micro-level architecture of the machine. The concept is illustrated graphically in Figure 1-4.

```
MACRO-LEVEL ARCHITECTURE
┌──────────────────────────────────────────────────────────────┐
│ PDP-11 Insturction set, General Registers, etc.               │
│ Programs reside in main memory                                 │
│                                                                │
│                                                                │
│          MACRO-LEVEL ORGANIZATION = MICRO-LEVEL ARCHITECURE    │
│        ┌──────────────────────────────────────────────────┐   │
│        │ 11/60 registers (≈100) and operational capabilites.│  │
│        │ Programs reside in control store                   │  │
│        │                                                    │   │
│        │                                                    │   │
│        │          MICRO-LEVEL ORGANIZATION                  │   │
│        │        ┌────────────────────────────────────┐     │   │
│        │        │                                     │     │   │
│        │        │                                     │     │   │
│        │        │                                     │     │   │
│        │        │           Hard-wired logic          │     │   │
│        │        │                                     │     │   │
│        │        │                                     │     │   │
│        │        └────────────────────────────────────┘     │   │
│        └──────────────────────────────────────────────────┘   │
└──────────────────────────────────────────────────────────────┘
```
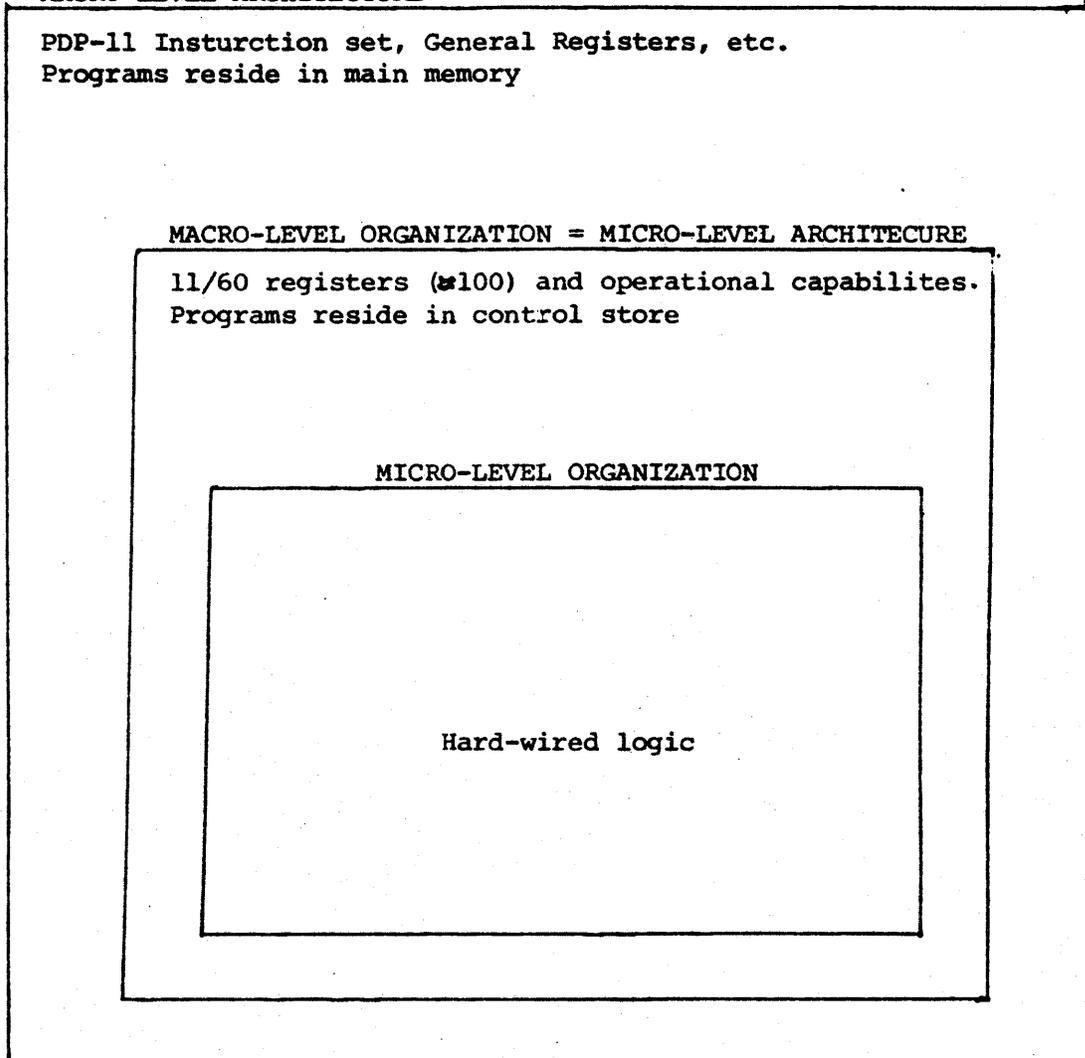
Figure 1-4 Hierarchical Structure of Memories,

Architecture, and Organization

1.2 THE 11/60 PROCESSOR

The 11/60 is a mid-range PDP-11 processor. It is a microprogrammed

implementation of the standard PDP-11 architecture. A floating

point unit, cache memory, stack limit, and memory management are

integral parts of the processor. With the Writable Control Store
(WCS) option, the user can augment the architecture of the PDP-11.

The micro-level architecture of the 11/60 is radically different
from the standard PDP-11 architecture, i.e., structure, visible
to the macro-level programmer. To successfully microprogram the
11/60, you must familiarize yourself with the details of its
micro-level architecture.

The 11/60 can be divided into five logical sections, as shown in
Figure 1-5. The microprogrammer's task is to control the flow
of data within each of these five basic sections, and sometimes
between them. Of primary importance is the Datapath section, where
most data handling functions are performed. The Datapath is
described in detail in Chapter 2.

Each section will be discussed in more detail later in this manual;
for the moment, it is only necessary to be aware of their general
function.

The Bus Control section contains the Unibus control logic, the
timing generator, and the console interface.

The KT/Cache section contains the memory management logic (KT); the
stack limit register (KJ) and 1024 words of high-speed cache
memory.

The Processor Control section contains the control store for the
base machine in the form of a read-only memory, or ROM; other
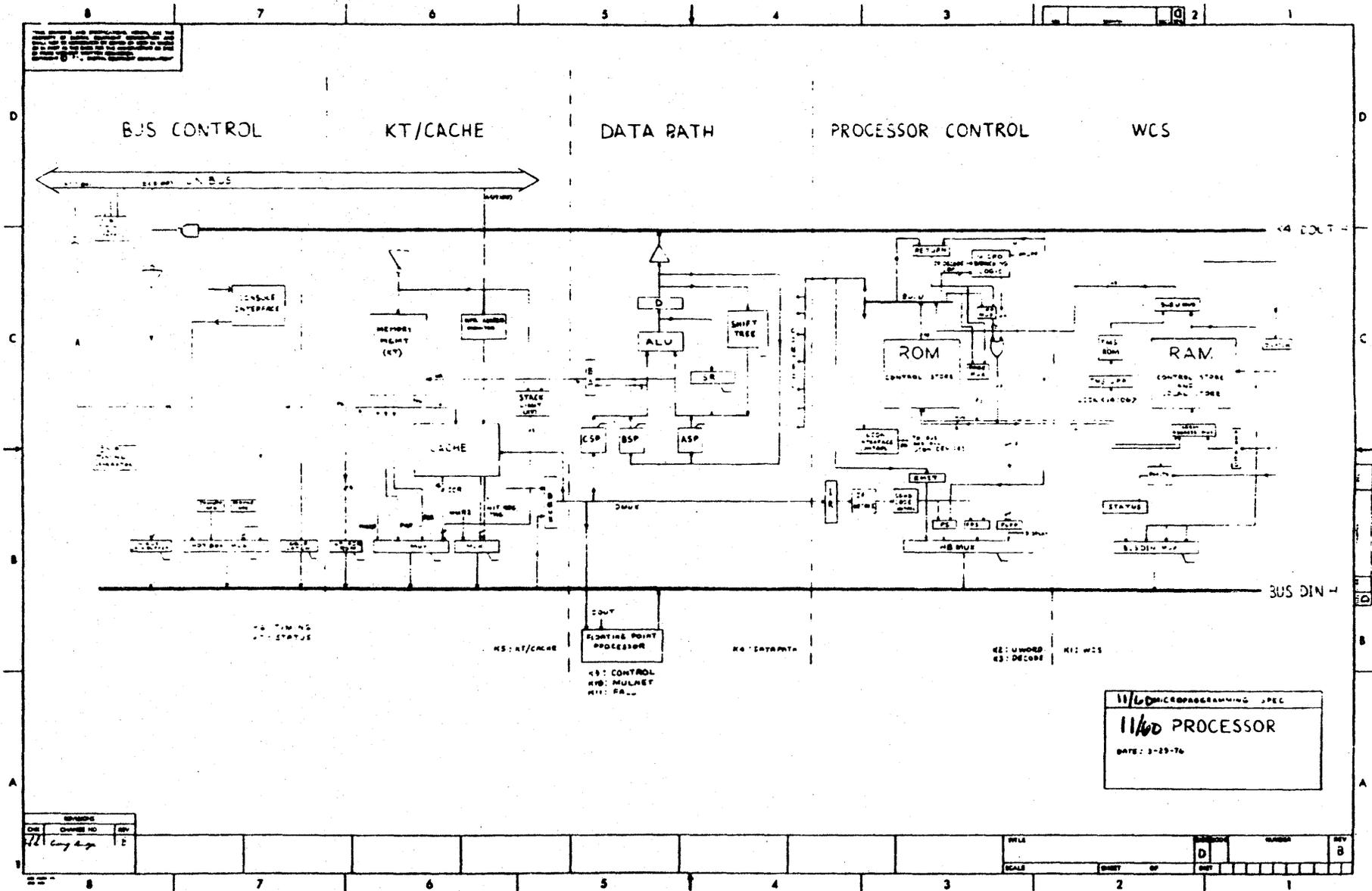
FIGURE 1-5   LOGICAL SECTION OF 11/60

control logic, the Processor Status word (PS) and the Floating

Point Status register (FPS).

The WCS section contains additional control store for the user

microprogrammer in the form of a RAM (Random Access Memory).

This RAM can also be used as a high-speed local store with the

aid of routines stored in the Transfer Micro store (TMS) ROM.

The main entry into the Writable Control Store is initiated by the

XFC.USER opcode, 0767xx. This PDP-11 machine instruction causes

control to be transferred to a special location, entitled

USERDISPATCH, in the WCS RAM.

1.3 THE USER CONTROL STORE OPTION

The principal use of the 11/60 microprocessor is the implement-

ation of the PDP-11 instruction set. However, the processor has

been designed with a dynamic control structure so that other

functions can be implemented. The UCS option provides additional

and alterable control store for the 11/60, enabling you to

extend the capabilities of your PDP-11. Possible applications

range from extending the PDP-11 instruction set to emulating a

computer with a different instruction set.

1.3.1 THE UCS PRODUCT

The Writable Control Store is a one-board hardware option for the

11/60 central processor, which includes a 1K-by-48 bit Random

Access Memory (RAM). This hardware by itself is not the complete

product.

To use the WCS hardware, that is, to do microprogram development
and debugging, DIGITAL provides the following software tools:
                    A MICRO-ASSEMBLER
                    A LOADER
The software tools for the WCS option are described in the UCS
Software Tools Reference Manual.

1.3.2 APPLICATIONS OF WCS

By design, the PDP-11 is a general-purpose computer. Thus there
are special-purpose computers which will perform better than will
a PDP-11 on those applications for which they have been designed.

WCS enables you to tailor, or bias, the PDP-11 to your particular
special-purpose needs. Such tailoring can be classified hierarch-
ically as follows.

Class 0 - Instruction Set Extensions
          Some functions were considered too special-
          purpose to be included in the original PDP-11
          design. These functions, such as block move
          and decimal arithmetic, can become new
          PDP-11 instructions. Their definition should
          conform to 11-instruction format and style.

Class 1 - Application Kernels
          Most applications and systems programs have
          sections which are executed much more
          frequently than others. A useful rule of
          thumb is that 10% of the code is executed
          90% of the time. Kernels within these critical
          sections can be microprogrammed for better
          throughput. Examples include the Fast Fourier
          Transform, an operating system's memory
          allocation routine, and Cyclic Redundancy Check
          calculation.

Class 2 - Emulation

> The interpretive execution of an instruction set
> by software is generally called simulation. When
> this interpretation is done by hardware it is
> called emulation. Microprogramming provides a
> means for inexpensively emulating several different
> instruction sets on one piece of hardware. The
> tasks involved in emulation include instruction
> decode, address calculation, operand fetch, and
> I/O operation as well as instruction execution.

Class 0 applications are relatively simple and straightforward uses
of microprogramming. Class 1 applications require more intensive
study and possibly statistical analysis if they are to improve
performance significantly.

The final class of applications, emulation, is best served by a
machine specifically designed as a general purpose emulator. The
11/60 was designed to emulate a PDP-11; hence, the organization of
its datapath is keyed to the 16-bit PDP-11 word and other
characteristics of a PDP-11 computer system. These factors in
large part determine what other computers can be emulated by the
11/60.

1.3.3 EXTENDED CONTROL STORE

To Be Supplied

1.4 USER INVESTMENT REQUIRED

To gain real benefit from use of the UCS option, you should invest
time and resources in two areas of study prior to attempting any

any WCS microprogramming. These two areas are: 1) understanding the 11/60 and 2) analyzing your proposed application.

1.4.1 DETAILED UNDERSTANDING OF THE 11/60

To microprogram the 11/60 effectively, you must study the internal details of the microprocessor--particularly the datapath. Although this is not a difficult task per se, users with little previous hardware exposure may have some problems in becoming accustomed to the hardware terminology and the notation used for hardware description. Moreover, the largely unprotected nature of the microprogramming environment may seem overly complex and unpredictable.

This manual discusses the 11/60 hardware at the functional level. Occasional references are made to the Engineering Drawings for the 11/60 (order no.): these references are provided only for those users whose curiosity would naturally lead them to the print set. Most users should find that this manual, used in conjunction with the UCS Tools Reference Manual, is all that is required to microprogram the 11/60 UCS effectively.

Appendix B of this manual contains a selective annotated bibliography of recent work on microprogramming and emulation.

1.4.2 DETAILED ANALYSIS OF PROPOSED APPLICATION

Of the three classes of microcode use described in Section 1.3.2., Application Kernels are the most likely "end-user" use of the Writable Control Store. Careful analysis is warranted.

Use of microprogramming will not always result in significant performance gains. Applications well-suited to microprogramming may improve performance by a factor of 5 to 10; poorly suited ones not at all. You must understand your application and analyze the execution of its individual instructions. This section is aimed at helping such analysis, but it is in no way a complete treatment of performance analysis.

A machine-language instruction goes through the following processing phases:

I-phase

Instruction fetched from memory and decoded.

O-phase

Operand addresses calculated; operands fetched from memory.

E-phase

Operation executed upon operands.

Each of these phases takes one or more microcycles. The total execution time, assuming no overlap of the phase, is the sum of these microcycles. Each phase can be seen as a candidate for elimination or for cycle-reduction through microprogramming, with resulting gains in performance.

The following generalizations can be made.

COMPOSITE OPERATIONS SAVE I-CYCLES

A block move on the PDP-11 can be programmed as:

```
          MOV COUNT, RØ           ; INSTRUCTION 1
          MOV #A, R1              ; 2:  FIRST SOURCE ADDRS TO R1
          MOV #B, R2              ; 3:  FIRST DESTINATION ADDRS TO R2
LOOP:     MOV (R1)+, (R2)+        ; 4:  MOVE AND INCREMENT BOTH ADDRS
          SOB RØ, LOOP            ; 5:  DECREMENT AND TEST COUNTER
```

Combining these operations into one instruction,

```
          BLOCKMOV #A, #B, COUNT
```

eliminates I-cycles, with the predominant savings coming from instructions four and five.

USING PROCESSOR STORAGE SAVES O-CYCLES

The microprogrammer can use internal CPU storage (the hardware registers) for intermediate results. There are a number of hardware registers, in addition to the general registers RØ-PC, which can be used by the microprogrammer to avoid memory cycles.

Because there is more parallelism at the micro-level, the inner machine (the microprocessor) is potentially more efficient than the outer machine (the PDP-11). Moreover, the microbranching logic structure of the microprocessor provides a broader decision logic capability which can be exploited, for example, in table search and string-edit operations.

In general, most cycle reductions which result from microprogramming come for the I- and O-phases of instructions.

When analyzing instuctions, you must also consider the ratio of
the time used by the I- and O-phases to that of the E-phase:

$$\frac{I \ + \ O}{E}$$

In polynomial evaluation or vector scalar multiplication, for
example, the cycles saved by a composite instruction are a small
fraction of the overall execution time.

In summary, you should analyze your application to develop
candidate sections for microprogramming, then apply detailed
analysis to the instruction execution sequence within these
sections before coding a microprogram.

1.5 FUNDAMENTAL MICROPROGRAMMING PARAMETERS

This section gives an overview of several topics which represent
fundamental parameters of the microprogramming environment.

First, the 11/60 microword is described in general terms. Next,
the basis for later discussion of timing is laid by a description
of the microcycle. Finally, the central program flow of the base
machine is described, and related to the discussion of I-, O-, and
E-cycles in Section 1.4.2.

1.5.1 THE 11/60 MICROWORD

This section reviews the general concept of instruction formats as
a foundation for describing the format of the 11/60 microword.

Note that an 11/60 microinstruction is exactly equivalent to one word of control store. Thus, the terms microword and micro-instruction are interchangeable. In this manual, however, a slight distinction has been made in the interest of clarity. Microword is used as a generic term for a control store word. Micro-instruction is used when focussing upon the control exerted by a particular microword.

1.5.1.1 INSTRUCTION FORMATS -- An instruction, whether at the macro-level or the micro-level, is the basic mechanism that causes a procedure to be invoked. Instructions usually take two source operands and produce a single result. This kind of instruction has five logical functions:

1 and 2)   Specify the address (location in storage) of the two source operands

3)   Specify the address at which the result of the operation is to be stored

4)   Specify the operation to be performed on the two source operands

5)   Specify the address of the next instruction in the sequence.

These specifications may be explicit or implicit. Implicit specification saves space in the instruction at the expense of additional instructions in the sequence.

There are four common formats for instructions: three-address, two-address, single-address, and zero-address (stack-type). These categories indicate how many of the address specifications are

explicit in the instruction.

A normal PDP-11 instruction of the form OPR SRC DST uses a two-
address instruction format. The address of both the source
operands are explicitly specified. The result address is implicitly
specified by the address of the destination operand. The next
instruction to be executed is implicitly identified by the
contents of the Program Counter.

The 11/60 microword, on the other hand, uses a four-address
instruction format: two source operand addresses; result address;
and next instruction address are all explicitly identified in
each instruction. There is no microprogram counter analogous to
the PDP-11 PC.

1.5.1.2 SEQUENCING AND BRANCHING -- Because there is no
incremental program counter at the microprogramming level in the
11/60, each microinstruction specifies the address of its successor.
Therefore, there is no requirement that microinstructions execute
sequentially according to their storage address.

Moreover, each microinstruction can also specify a branch condition
to be tested before the next microinstruction is fetched. The
result of the test can cause a different microinstruction to be
fetched.

1.5.1.3 MICROWORD FIELDS -- The 11/60 microword is divided into
fields, each of which is associated with a particular functional
unit or control function. Not all fields are contiguous, and they

can overlap. That is, a single bit can be used to generate
more than one control signal.

The interpretation of some overlapping fields in the 11/60 micro-
word are controlled by a technique known as bit steering. A few
bits in the microword are set aside to specify how the bits in
other fields of the microword are to be interpreted.

For example, there are two bits in the 11/60 microword that can be
used either to control scratchpad writing or to clock registers on
the datapath. A third bit is used to specify what the first two
bits mean, as illustrated in Figure 1-6.
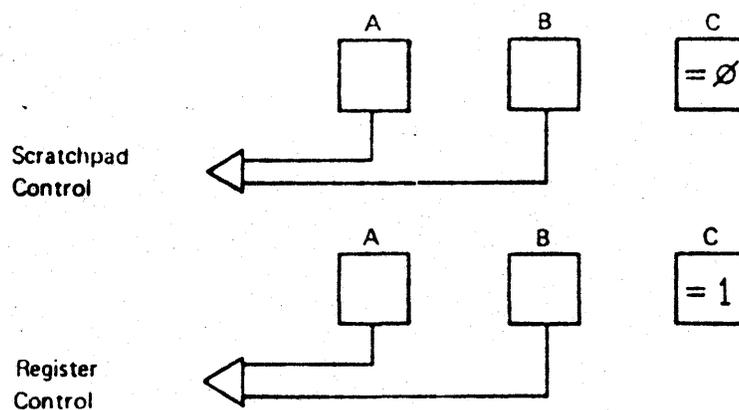
Figure 1-6 Bit Steering

Other cases where fields overlap are protected from conflicts
because the different uses of the same bits are mutually exclusive.
For example, the literal field overlaps the ALU function field. A
microinstruction which specifies a literal value will generally
specify operations to store that data correctly in the datapath.
Another microinstruction would manipulate that literal data.

1.5.1.4 WIDTH AND ENCODING OF THE MICROWORD -- The standard width

of a control store word on the 11/60 is 48 bits. There are

extensions in some sections of the base machine control store

which make the microword 56 bits wide. This manual will discuss

only the 48 bits available to the UCS user, because the 8 extension

bits are highly specific to PDP-11 emulation.

The 11/60 employs what is known as a "horizontal" microword. That

means that a majority of the bits in the microword are directly

used to generate some signal within the machine. Some of the fields

are encoded, meaning that the value represented by the bits in that

field must be decoded before control signals are generated.

The term horizontal also implies a significant degree of parallelism

within the 11/60 datapath. One microinstruction can, in some

circumstances, be much more powerful than one macro-level instruction.

Figure 1-7 is a summary diagram of the 11/60 microword. Most of

the notation will not make sense to you now, since each of the fields

will not be described in detail until later chapters. It will be

useful to refer back to this diagram from time to time to see

how the pieces fit together.

1.5.2 THE MICROCYCLE

Timing is extremely important to the microprogrammer. It imposes

constraints on the operations that can be done within one micro-

instruction, as well as what can be done within a group of micro-

instructions. An awareness of what happens when will help to avoid

trivial, but troublesome, errors.

```
 47                                          30  29      25  24     20  19        14  13                         0
┌──────────────────────────────────┐  ┌──────────┐  ┌──────────┐  ┌──────────┐  ┌──────────────────────────────┐
│      ALUF (BUSB, BUSA)           │  │  CLOCKS  │  │ BUS/BOX  │  │Scratchpad│  │        SEQUENCING            │
│                                  │  │          │  │ CONTROL  │  │ Rewrite  │  │                              │
└──────────────────────────────────┘  └──────────┘  └──────────┘  └──────────┘  └──────────────────────────────┘
```

Return Page   RETURN ADDRESS   PAGE

E   M   I   T
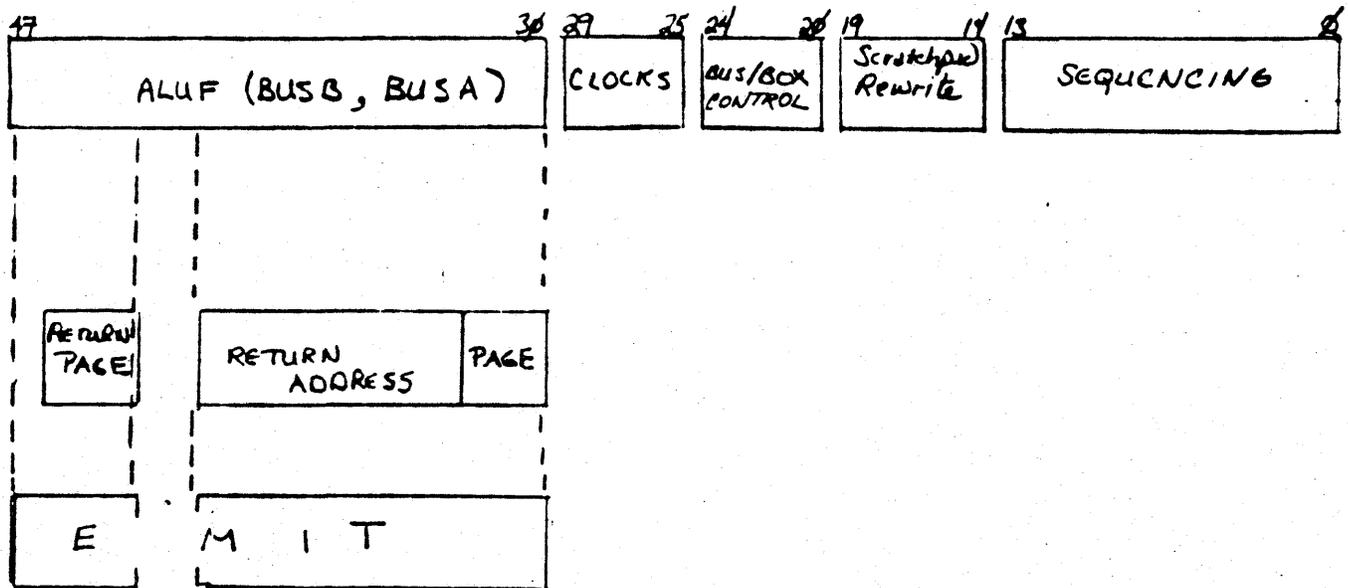
Figure 1-7 . Microword Summary

1-25

A new microword is given control of the 11/60 at the beginning of each processor cycle, or microcycle. This microword controls the activity on the datapath throughout that microcycle.

The 11/60 microcycle is 170 nanoseconds long. During this time, there are four clock pulses: P1, P2, P3, and μP3 (micro-P3). The microcycle is defined as the time period between two consecutive trailing edges of μP3.

The other pulses, P1, P2, and P3, control the timing of events on the datapath. You will primarily be concerned with the timing of register loading. Inputs to a register must be stable before the register is loaded, or invalid data will be stored.

For example, the result of an ALU operation can be loaded into a storage register at P2.

A microword, and the microcycle during which it is in control of the 11/60, is but one step in the execution of a PDP-11 instruction. Each of the three clock pulses P1, P2, and P3 further divide this step: a number of data transfers can occur during one 11/60 microcycle.

Figure 1-8 shows the relationship of the clock pulses to the microcycle.
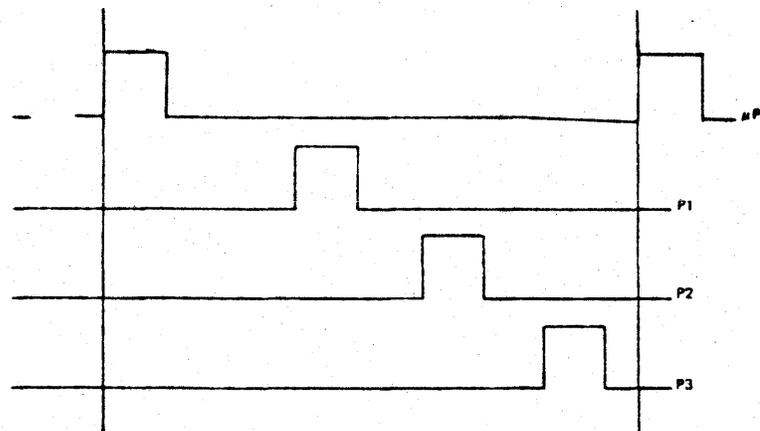
Figure 1-8 The Microcycle

1.5.3 MICROPROGRAM FLOW

The basic interpretive loop of instruction execution in 11/60 microcode is as follows:

```
  ┌──── FETCH   memory word addressed by PC
  │            │
  │            ▼
  │       INCREMENT PC
  │            │
  │            ▼
  │        DECODE
  │            │
  │            ▼
  │        EXECUTE
  └──────────▲
```
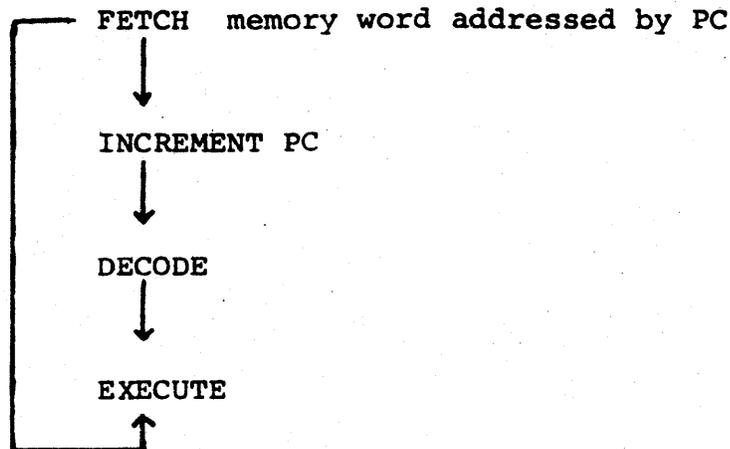
Every microprogram invoked by a PDP-11 opcode follows this pattern. The instruction currently pointed to by the contents of the PC is brought into the processor from main memory and stored in the Instruction Register, or IR. The PC is incremented by two so that it points at the next location to be accessed. The decode step identifies what instruction is to be executed, and dispatches control to the proper section of microcode. After the operation is performed, another instruction is fetched.

A slightly more detailed flow structure is shown in Figure 1-9. Note that at the completion of the instruction execution, a test is made for service conditions. If no service condition, such as an interrupt, exists, the next instruction is fetched. If a service condition does exist, control passes to another micro-
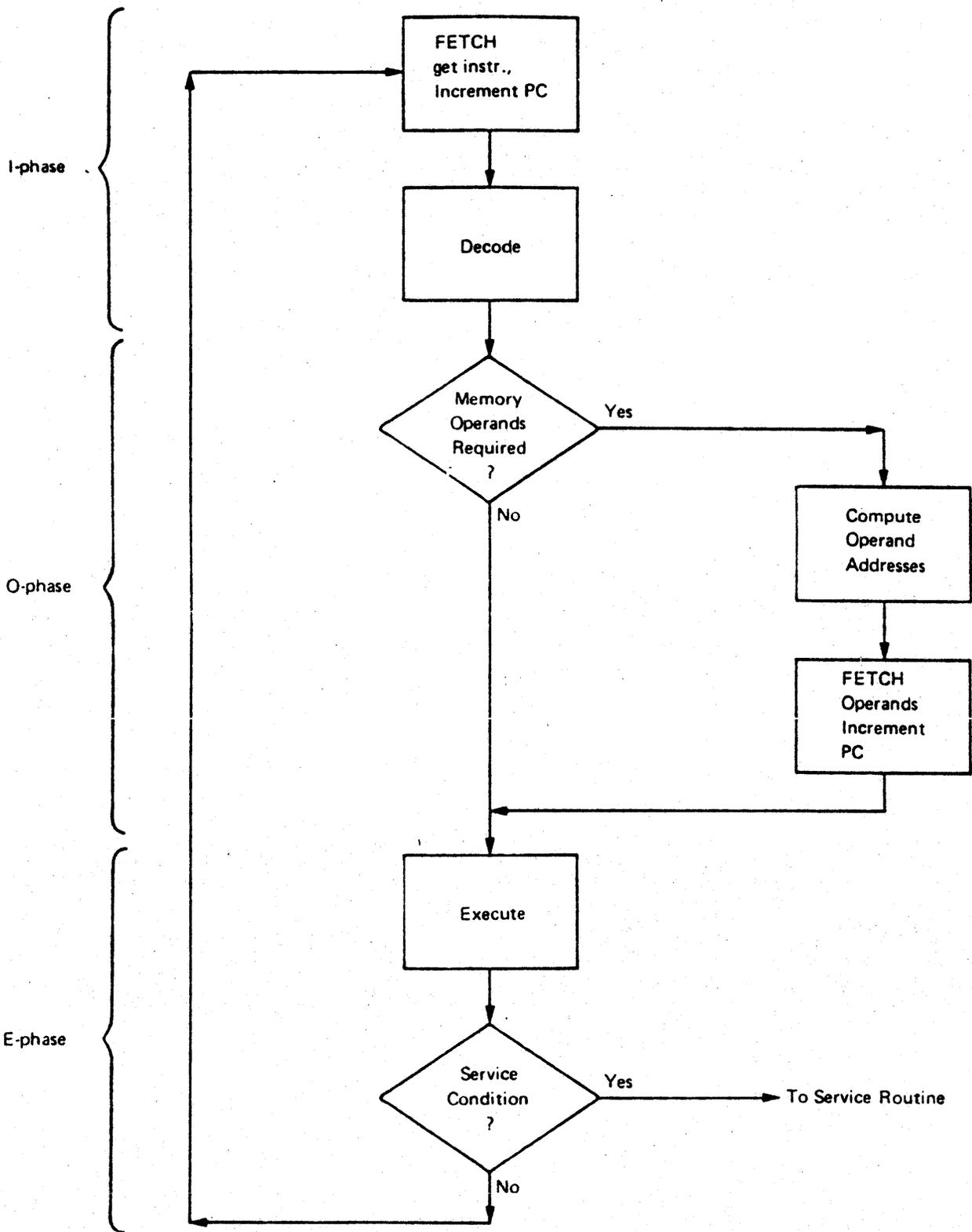
FIGURE 1-9 Program Flow in the 11/60

1-28

program which handles the interrupt or other condition. The I-, O-, and E-phases are noted at the left side of the diagram.

1.6 STRUCTURE OF MANUAL PRESENTATION

Two aspects of the 11/60 hardware are of prime concern to the microprogrammer: the data flow and the control flow, or control structure.

There are three distinct kinds of data flow in the 11/60:

Data flow within the datapath

Data flow within the inner machine

Data flow to the rest of the world.

This data flow implies the model of the 11/60 in Figure 1-10. This model provides a different logical structure from that presented in Figure 1-5; this manual uses this new model as a conceptual framework for the discussion of the 11/60 hardware.

The microprogrammer's world is the Inner Machine: the datapath and processor control sections of the processor. There are three interfaces between the Inner Machine and the rest of the computer system: data in, data out, and address out.

This manual focusses on the Inner Machine and the microprogramming techniques for controlling it. Becuase these two major topics are interrelated, and because both must be understood before you can microprogram the 11/60, this manual discusses them in parallel.

The data flow within the 11/60 datapath is described in Chapter 2, with minimal reference to other parts of the model.

Chapter 3 introduces the control structure of the 11/60, and discusses timing considerations. Further details are contained in Chapter 5.

Chapter 4 extends the discussion of data flow to the inner machine, and then to the rest of the CPU.

The material in Chapter 2, 3, and 4 is highly interdependent. One result is the Chapter 2 may seem overly detailed until you have finished reading Chapter 4.

Similarly, the UCS Usage Guidelines and the Examples have been placed at the end of the manual so that they may be discussed in the context of previously presented information.
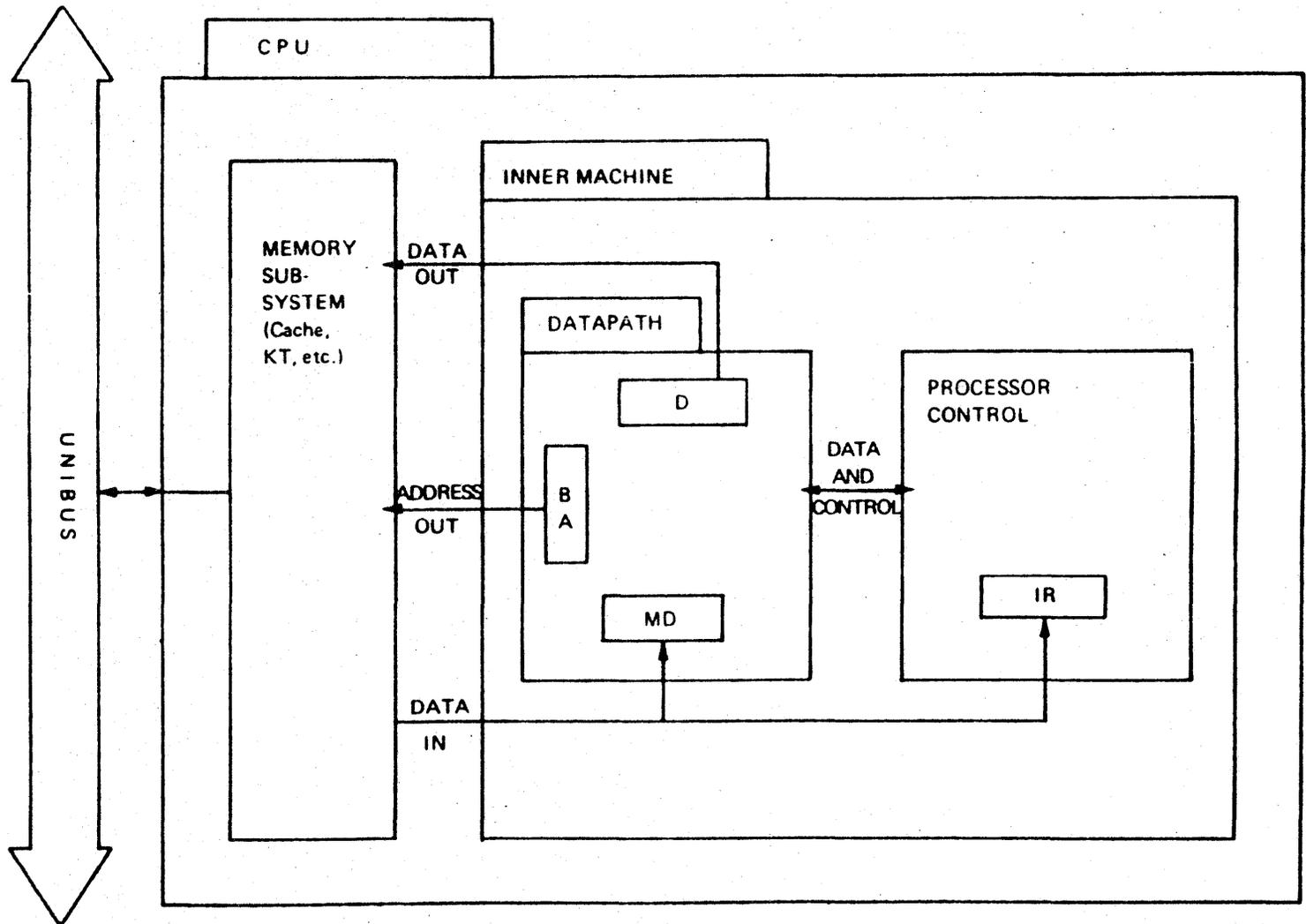
Figure 1-10

CHAPTER 2

THE 11/60 DATAPATH

DRAFT

The datapath[1] section of the 11/60 routes, manipulates, and
stores data within the processor.

This chapter describes the basic functional components of the
datapath and the corresponding control fields in the micro-
word. Looking at each component individually provides a
secure basis for understanding the relationship of the data-
path hardware to the overall problem of microprogramming the
11/60.

At the end of this chapter is a block diagram of the complete
datapath (Figure 2-37). As you read through the chapter,
refer to this fold-out diagram to see how the pieces fit
together.

2.1   THE HEART OF THE DATAPATH

The heart of the 11/60 datapath is the computational loop
shown in Figure 2-1.

There are two scratchpads (ASP and BSP), each connected to a
tri-state bus (BUS AIN and BUS BIN). These buses provide
input to the ALU. The other ALU input comes from the CIN
multiplexer, which provides the carry-in bit.

D is a 16-bit register which holds the output of the ALU.
This data can be directed back to the scrathpads after

---

[1] 17874

2-1

storage in D. D(C) holds the selected carry-out bit from
the ALU operation.
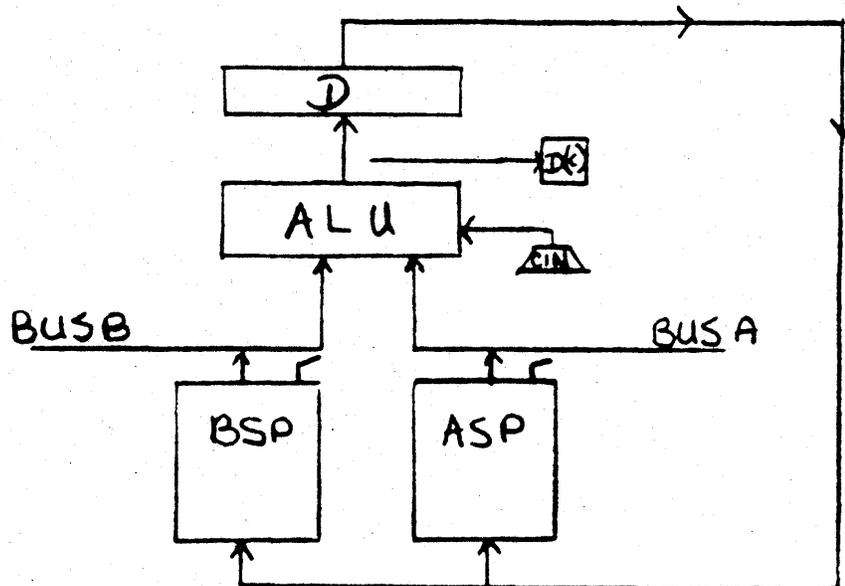
Figure 2-1:   The Heart of the Datapath

Control of the data flow among these components is provided
by the microword.

2.1.1   The ALU Field of the Microword

The ALU[2] receives two 16-bit words from BUS BIN and BUS AIN,
performs an arithmetic or logical operation upon them, and
produces a 16-bit result.

The operation performed by the ALU is determined by the ALU
field of the microword. This field occupies bits 47 through
44, which is represented as $\mu<47:44>$. Each of the sixteen
possible values of this field selects a unique ALU function[3].

---

[2]74S181 in semiconductor vendors' catalogs.

[3]The function codes shown in a vendor's catalog for the 74S181
  are not the same as the codes used in the 11/60 μword.
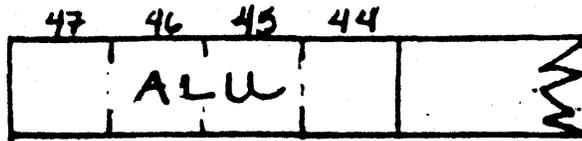
Figure 2-2: ALU Field of the μword

Table 2-1 shows the function invoked by the various values
of the ALU field and the corresponding source for the carry-in
bit. (The carry-in is described in detail in Section 2.1.5).

TABLE 2-1

ALU CONTROL FIELD ENCODING

OCTAL VALUE	VERBAL DEFINITION	CIN SOURCE
0	Complement A	1
1	A plus B plus PS(C)	PS(C)
2	(NOT A) and B	PS(C)
3	Generate 0	PS(C)
4	A plus B plus D(C)	D(C)
5	A plus (NOT B) plus D(C)	D(C)
6	A Exclusive OR B	D(C)
7	A AND (NOT B)	D(C)
10	Subtract B from A if D(C) = 1 Add if D(C) = 0	0
11	A plus B	0
12	Select B	0
13	A AND B	0
14	A plus B plus 1	1
15	A minus B	1
16	A Inclusive OR B	1
17	Select A	1

Notice that ALU operations such as A plus B plus PS(C) and
A plus B plus D(C) serve the same function as PDP-11
instructions like ADC, without requiring a separate
micro-instruction for handling the carry.

2.1.2   The B and A Scratchpads

Primary data storage for the datapath is provided by the
A and B scratchpads, each of which contains 32 registers.

Each of these scratchpads is divided into two sections of
16 words each; a HI section and a LO section: (refer to
Appendix B for an explanation of the notation)

            BSPLO :=   BSP[0:17]<15:00>
            BSPHI :=   BSP[20:37]<15:00>

            ASPLO :=   ASP[00:17]<15:00>
            ASPHI :=   ASP[20:37]<15:00>

BSPLO and BSPHI have separate outputs onto BUS BIN; similarly,
ASPLO and ASPHI have separate outputs onto BUS AIN, as shown
in Figure 2-3.
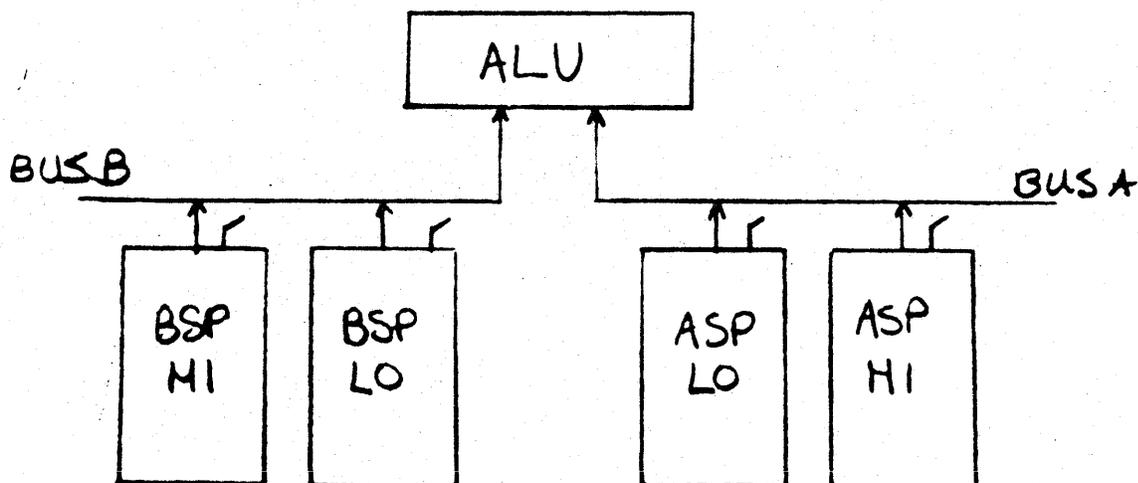
Figure 2-3: BSP and ASP

2.1.3   The D Register

The purpose of the D register is to store the ALU output, either
for testing or for routing elsewhere in the datapath or the
processor. The ALU result can be clocked into D either at P2
or P3. (When a register is clocked, the data at its input is
immediately transferred to its output; the output does not
change until the register is clocked again or cleared.)

Two fields in the microword, each one bit wide, control the
D register. CLKD, μ<28>, specifies whether or not D will be
loaded in the current microcycle. The time at which D is
clocked is determined by the WHEN field, μ<29>.

The D register is clocked only if the CLKD field contains a 1.
If WHEN contains a 0, clocking occurs at P2; if WHEN contains
a 1, clocking occurs at P3.
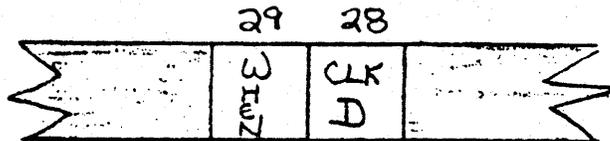
Figure 2-4:   WHEN, CLKD Fields

After an ALU result has been clocked in the D register, it can
be directed to a variety of places: other datapath logic;
other sections of the processor; main memory; or temporary
storage in the scratchpads.

2-5

2.1.4   Multiplexers

A multiplexer is a component which has several data input ports
and only one output. Selection signals control which input
port's data is gated to the output. Data is neither modified
no stored when it passes through a multiplexer.

Both the input ports and the selection signals for a multiplexer
are numbered. The (control) data at the selection ports forms
a binary number which designates one input port.

For example, a four-to-one multiplexer, as shown in Figure 2-5,
has two selection signals, S0 and S1. There are four input
ports, A, B, C, and D; where A is the low-order, or 0, port.
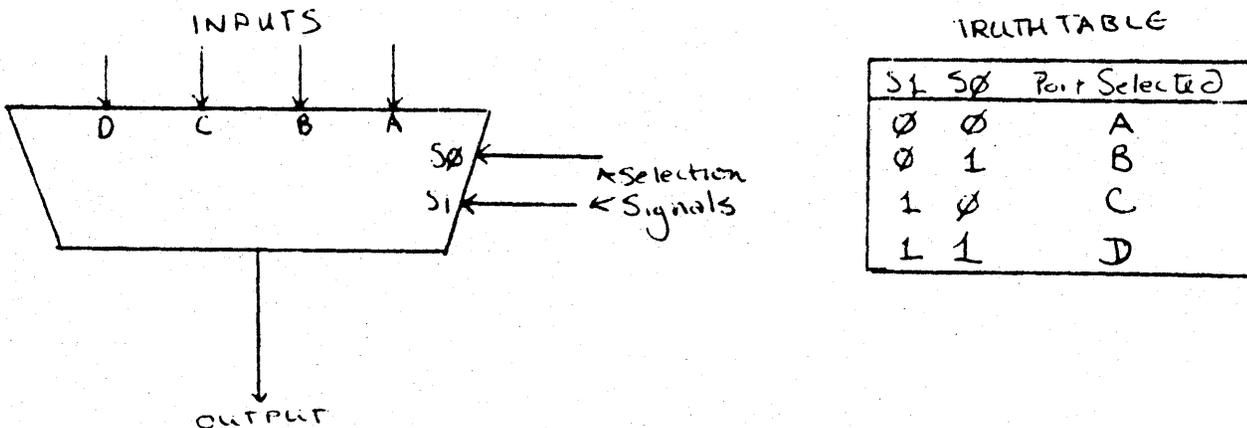
Figure 2-5: Four-to-One Multiplexer

If S0 and S1 are both 0, then the data at port A is transferred
to the output of the multiplexer. If S1 is 1, and S0 is 0, then
the C port is selected. The truth table in Figure 2-5
illustrates this correspondence.

2.1.5   ALU Carry Bits, CIN and D(C)

Section 2.1.1 described ALU function control and mentioned
the carry-in bit, CIN. This section examines both the carry-
in and carry-out bits of the ALU and their relationship to
each other. Both CIN and the carry-out bit D(C) are selected
by multiplexers. The multiplexer which selects the CIN bit
has four inputs: 0, 1, PS(C) (the C-bit of the PSW), and
D(C), the last carry-out. Selection of this multiplexer is
controlled by the ALU function code.

After an ALU operation is complete, the 16-bit result can be
clocked into the D register. If the D register is clocked,
D(C) is clocked at the same time. The bit which becomes D(C)
may be the actual carry, or overflow bit of the ALU; hence
the term carry-out is used. However, the overflow is not the
only source for D(C).

It is best to consider D(C) as a state bit retained from an
ALU operation - sort of an internal condition code. It has
a number of different functions. As the carry output of the
ALU, D(C) can be fed back into another ALU operation through
CIN, thur providing a facility analogous to the PDP-11
operations ADC and SBC. D(C) is also used to load the C-bit
of the Processor Status Word, and is also used as a test
condition for microcode branching.

The source for D(C) is chosen by the COUT MUX. Unlike the
multiplexer for CIN, the COUT MUX is controlled directly
from the microword. Indirectly, this does affect CIN
selection when the ALU function of the next microword uses
D(C) as the CIN source. In these cases, the selection for
the COUT MUX in one instruction will determine the source
for the CIN bit in the next microinstruction.

Figure 2-6 shows the relationship between the ALU, the CIN MUX, and the COUT MUX.
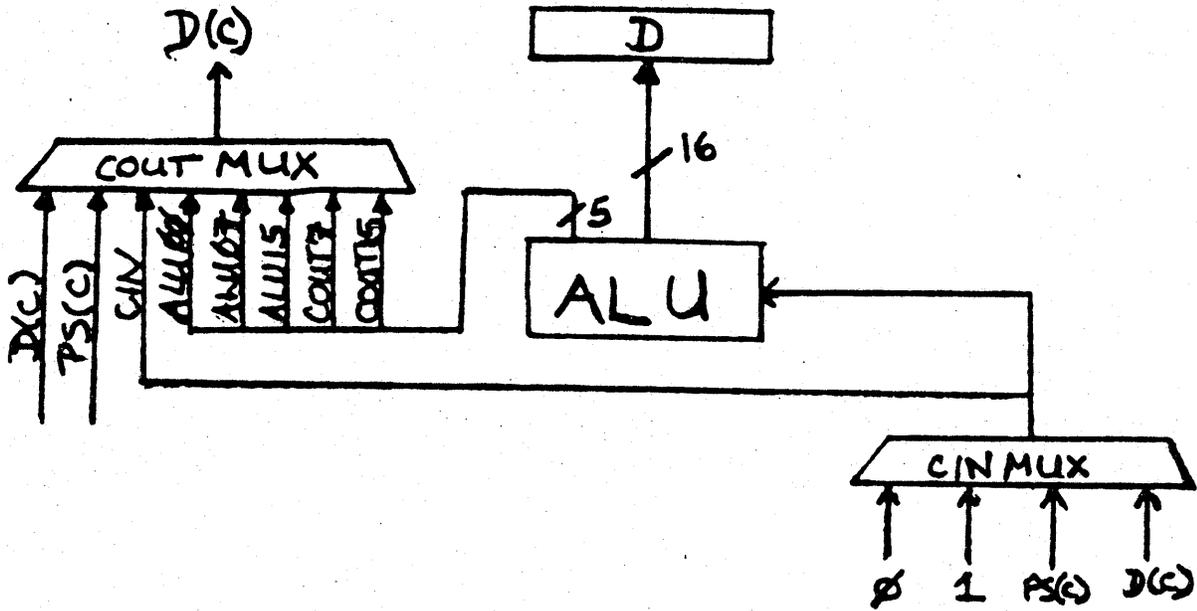
Figure 2-6: CIN, COUT of ALU

2.1.5.1   Selection of D(C) Source

COUT07 and COUT15 are, respectively, the byte and word carries from the ALU operation. This carry bit can then be clocked into the Processor Status word, PS, or fed back into a subsequent ALU operation. For example, during a 32-bit add, the carry-out bit from the addition of the low-order words becomes the carry-in bit for the addition of the high-order words. COUT∅7 and COUT15 are undefined when a logical operation is performed.

ALU15 is bit 15 of the ALU result, the sign bit. Testing for a negative result and some shifting operations would select this source for D(C).

ALUØ7 is bit 7 of the ALU result, which is the sign bit of a byte quantity.

ALUØØ is the Ø bit of the ALU, which indicates an odd or even result.

CIN is the output of the CIN MUX, the same carry-in bit presented to the ALU. This allows you to select a 1 or a 0 for D(C) directly, depending on the ALU code.

PS(C) is the C bit of the Processor Status word; the base machine uses it as the D(C) source for those PDP-11 instructions in which the C bit of the PSW does not change.

D(C) is the D(C) bit generated by the previous CLK D specification. This allows you to save, or recycle, a D(C) value from the last time an ALU result was clocked into D.

2.1.5.2   Control of COUT MUX -- The COUT MUX is controlled by the COUT field of the microword, $\mu<32:30>$.
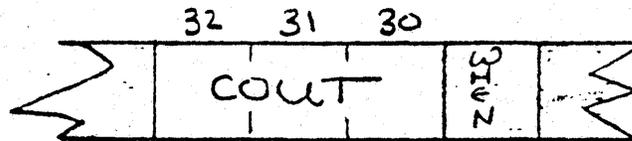
Figure 2-7: COUT Field of Microword

The encoding of the COUT field of the microword is shown in Table 2-2.

Note that, regardless of COUT, D(C) is not changed unless
CLK D = 1; D(C) is clocked at the time specified by the
WHEN field.

Table 2-2
COUT FIELD Encoding

D(C) SOURCE	MNEMONIC	COUT FIELD VALUE
Output of CIN MUX	CIN	0
C bit of the PSW	PS(C)	1
Bit 0 of ALU result	ALUØØ	2
Bit 7 of ALU result	ALUØ7	3
Bit 15 of ALU result	ALU15	4
Byte Carry	COUT7	5
Word Carry	COUT15	6
Carry-out from previous operation	D(C)	7

2.1.6   Setting the Condition Codes

The condition codes, N, Z, V, and C of the Processor Status
Word, are macro-level state indicators whose values are defined
for every PDP-11 instruction. Their purpose is not to record
the status of the micro-level machine after every microcycle,
and hence these bits are clocked only when specifically
indicated by the microprogrammer.

There are two ways to set the condition codes; only one of them
will be discussed here. A second, more general method is
described in Chapter 4.

The Set Condition Codes (SCC) field, μ<25>, controls the
loading of the PDP-11 condition codes. When SCC contains
a one, the condition codes are altered during the <u>next</u>
micro-cycle. D and D(C) must be clocked at P2 for the
condition codes to be set correctly. This timing relation-
ship is illustrated in Figure 2-8.

←———————— μcycle n ————————→ ←———— μcycle n + 1 ————————→

μword: CLKD/YES, WHEN/P2,
               SCC/YES, COUT/ALU15

action: P2, D ← ALU   P2,   PS(C) ← D(C)
             P2, D(C) ← ALU15   P2,   PS(N) ← D<15>
                                            P2,   PS(Z) ← D<15:00> = Ø
                                            P2,   PS(V) ← Ø
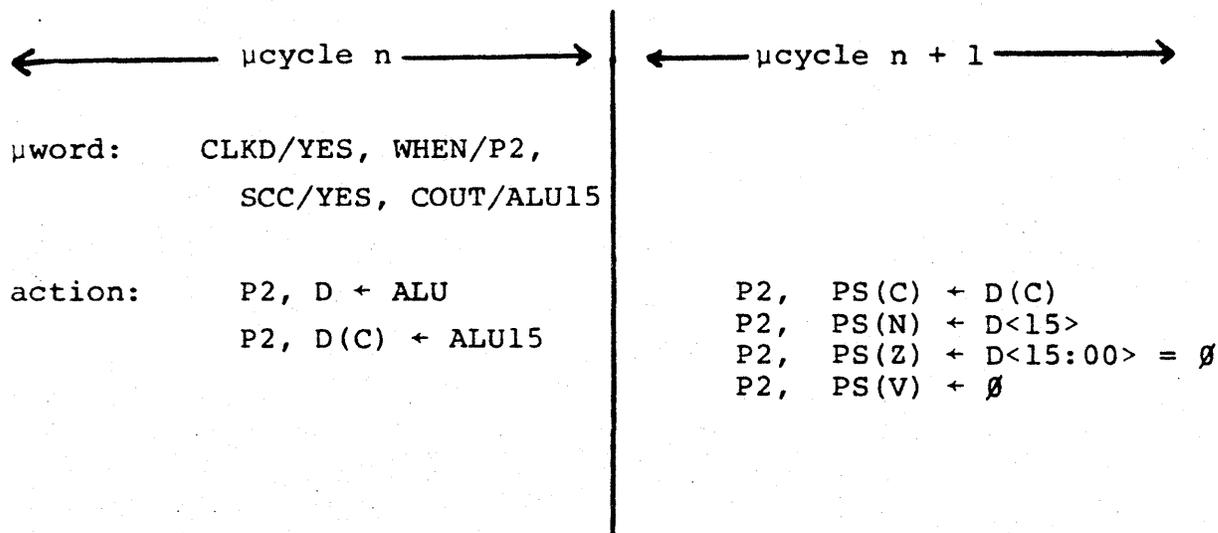
Figure 2-8: Condition Code Clocking

If the IR contains an XFC or other reserved opcode, then the
PDP-11 condition codes are clocked as follows. The C bit of
the PSW is loaded with D(C). (From the previous discussion of
D(C), you can see that there are actually eight sources for the
PS(C).) The N and Z bits reflect the status of the D register
at P2 of the microinstruction in which SCC was set. The V bit
is loaded with 0.

D and D(C) must remain stable through P2 of the microcycle
following the SCC/YES specification.

Whether or not you, as a WCS user, set the contition codes
during a microcycle depends on the requirements, or
expectations, of the macro-level program. For example, if
your macro-level program needs to branch upon conditions
resulting from an XFC instruction, you would clock the
condition codes.

2.2   BUS BIN AND BUS AIN

The buses that provide operand input to the ALU are tri-
state buses; that is, they connect a number of tri-state
devices. The use of tri-state logic in the 11/60 allows
a multiplexing function to be performed without actually
using a multiplexer, with resulting hardware savings.
The symbol Γ denotes a tri-state device.

A number of sources on either side of the ALU can be
selectively enabled onto BUS BIN or BUS AIN. Figure 2-9
shows the relationship of the ALU input sources to the
portion of the datapath previously discussed.

On the B-side of the ALU, there are three sources: the
two sections of the BSP, and another 16-location scratchpad,
the CSP. On the A-side, there are four locations: ASPLO,
ASPHI, the XMUX, and the Shift Tree. Each of these components
will be described in detail in succeeding sections.

The BEN field of the microword, $\mu<43:42>$, controls which
source is enabled onto BUS BIN; the AEN field, $\mu<39:38>$,
determines which source is enabled onto BUS AIN. Table
2-3 defines the encoding of these fields. Two BEN codes
are dedicated to the CSP because there are two methods of
providing addresses to this scratchpad.
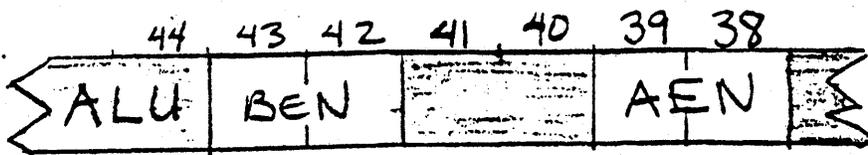
Figure 2-9: BEN, AEN Fields of Microword
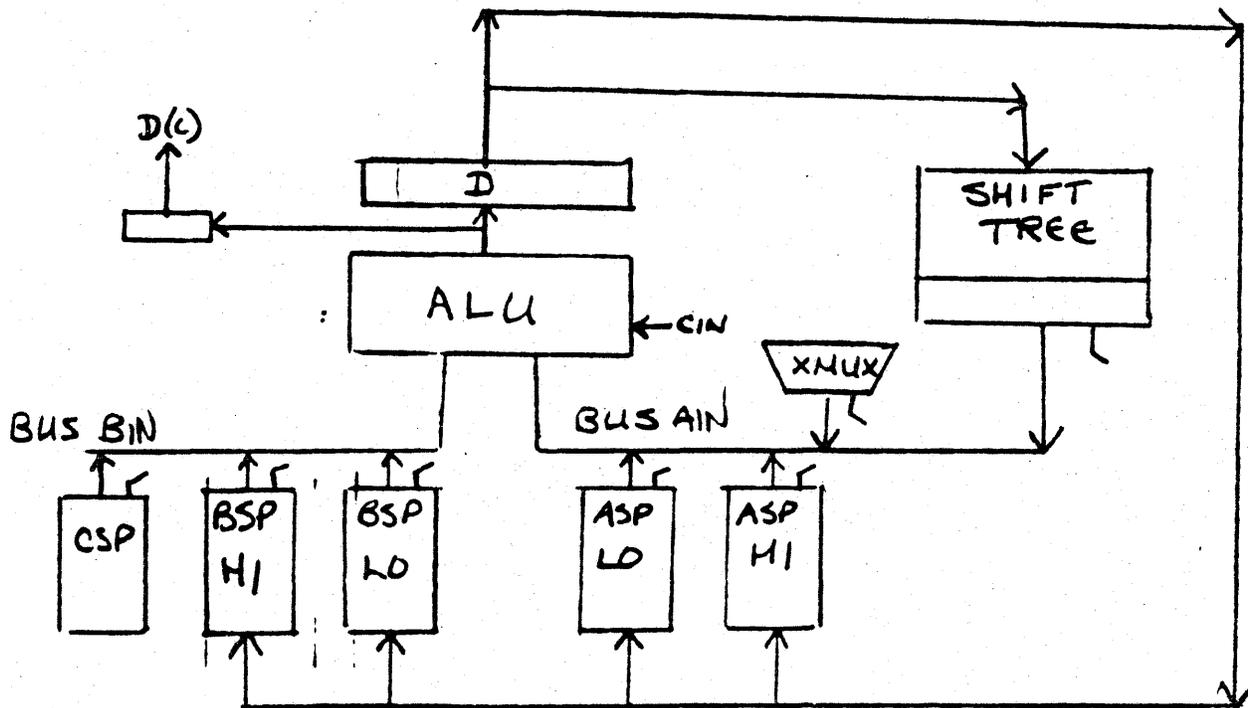
Figure 2-10: BUS BIN and BUS AIN Sources

Table 2-3

Bus Enable Field Encoding

BUS B			BUS A		
SOURCE ENABLED	MNEMONIC	BEN VALUE	SOURCE ENABLED	MNEMONIC	AEN VALUE
BSP[0:17]	BSPLO	0	XMUX	XMUX	0
BSP[20:37]	BSPHI	1	Shift Tree	CMUX	1
Arbitary CSP location	CSP	2	ASP[0:17]	ASPLO	2
Base Constants	BASCON	3	ASP[20:37]	ASPHI	3

2.2.1   Organization of ASP and BSP

The organization of the B and A scratchpads is shown in
Figure 2-10.

The first eight locations of ASPLO and BSPLO are reserved for
the PDP-11 general registers R0-PC. These registers are
duplicated to allow concurrent access of two registers. This
allows register-to-register operations to be performed in a
single microinstruction. The User Stack Pointer is duplicated
in BSP[16] and ASP[16].

Three locations are reserved for the WCS user; these are
indicated in the illustration as WCSB[n] and WCSA[n]. The
contents of these registers is not altered by any of the base
machine or floating point microcode.

The standard microcode floating point implementation uses
ASP [10:15 , 30:35] and BSP [10:15, 30:35] as the floating
point accumulators. If the FP11-E floating point processor
is present, these locations are also available for the WCS
user. No other standard microcode uses these registers.

The remaining registers fall into two classes: those which
the WCS user may alter, and those which you must not alter.

BSP[0:17] BSPLO	BSP[20:39] BSPHI	ASP[0:17] ASPLO	ASP[20:37] ASPHI
R0	WCSB[0]	R0	WCSA[0]
R1	WCSB[1]	R1	WCSADR
R2	R(VECTSAV)	R2	GEN-WHAMI
R3	R(ZERO)	R3	CNSL.TMRSW
R4	R(T4B)	R4	R(T1A)
R5	R(T2B)	R5	R(T2A)
R6	FPA	R6	CNSL.SW
R7	CNSL.CNTL	R7	CNSL.ADR
FAC2[0]	FAC0[0]	FAC3[0]	FAC1[0]
FAC2[1]	FAC0[1]	FAC3[1]	FAC1[1]
FAC2[2]	FAC0[2]	FAC3[2]	FAC1[2]
FAC2[3]	FAC0[3]	FAC3[3]	FAC1[3]
FAC2[4]	FAC0[4]	FAC3[4]	FAC1[4]
FAC2[5]	FAC0[5]	FAC3[5]	FAC1[5]
USER R6	FEA	USER R6	FPSHI-FEC
FDST2	FDST0	FDST3	FDST1

Figure 2-11: BSP and ASP Layout

2-16

2.2.1.1   Temporary Storage Registers -- A WCS microprogram
can use the registers which the base machine and floating point
use for temporary storage during instruction execution.

The temporary storage registers used by the base machine are:

```
            BSPHI[4]   :=   R(T1B)
            BSPHI[5]   :=   R(T2B)
            ASPHI[4]   :=   R(T1A)
            ASPHI[5]   :=   R(T2A)
```

The state of these registers is not saved if the base machine
code is invoked. Thus, data stored in these registers may be
overwritten by the base machine microcode that handles error
conditions, or if a new macro-level instruction is fetched.

The following registers are used for temporary storage by the
floating point microcode and by the FP11-E.

```
            BSPLO[17]   := FDST2
            BSPHI[17]   := FDST0
            ASPLO[17]   := FDST3
            ASPHI[17]   := FDST1
```

User data stored in these registers will be lost if a floating
point instruction (17xxxx) is fetched.

2.2.1.2   Reserved System Registers -- The remaining 11
registers in the B and A scratchpads are used to store
console, status, address, and constant information. These
registers are reserved for use by the base machine and must
not be altered. They may, however, be read.

WCSADR, ASPHI[1], has the contents of Unibus address 177542.
It is used to specifiy an address within the WCS control store
space to which data is to be written. (See Chapter 6)

R(VECTSAV), BSPHI[2], contains the vector address of the last
interrupt serviced. This address is saved to aid error
diagnosis.

FPA, BSPHI[6], is used by the floating point microcode and the
FP11-D to hold the address (incremented by two) of the last
floating point instruction.

CNSL.CNTL, BSPHI[7], contains console control and status
information. It also contains the two high-order bits of the
switch register, the temporary switch register, and the console
address register.

FEA, BSPHI[16], contains the address of the last floating point
instruction that incurred an exception.

The WHAMI (What Am I) register, ASPHI[2], contains status
information for the micro-machine. Layout of the WMAHI
register is shown in Figure 2-1.
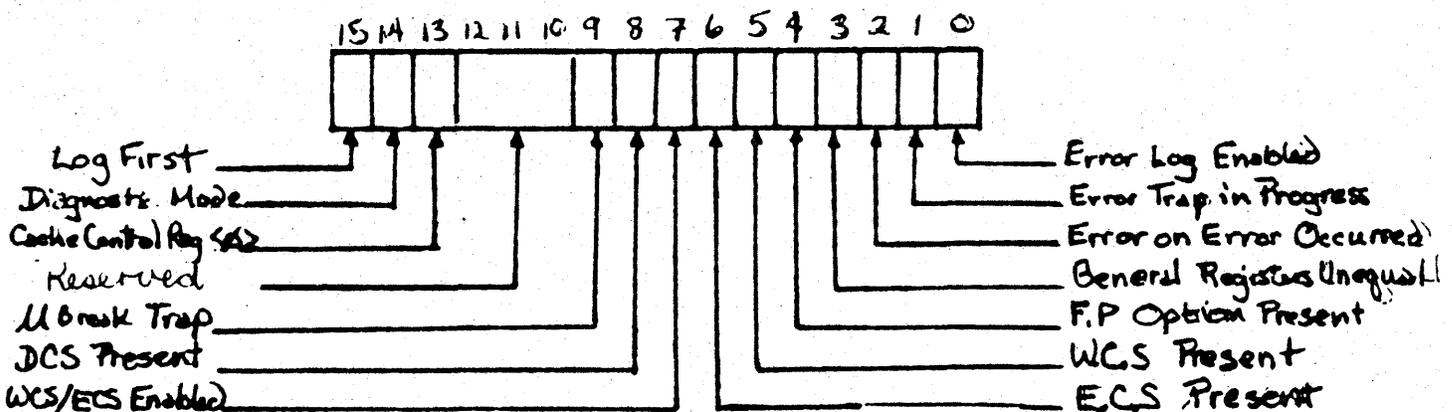
Figure 2-12:   WHAMI Register

CNSL.TMPSW, ASPHI[3], is used to assemble numbers from the console keypad before transfer to the switch register. It is also used in the display subroutine in the console microcode.

CNSL.ADR, ASPHI[7], is the console address register. It is loaded with CNSL.TMPSW data on LOAD ADRS. On moves to 777570, the data is loaded into CNSL.TMPSW before being displayed on the console.

The high byte of FPSHI-FEC, ASPHI[16], contains the high byte of the Floating Point Status Register. The low byte of FPSHI-FEC contains the exception code of the last floating point instruction that caused an exception.

R(ZERO), BSPHI[3], contains the value zero. It is used whenever a $\emptyset$ is needed from the B-side during a cycle in which the CSP is written. This location must always contain the value 0.

2.2.1.3   Integrity of the General Registers -- For the 11/60 to operate correctly, the scratchpad locations reserved for the PDP-11 general registers must contain those registers. The contents of the corresponding registers in ASPLO and BSPLO must be identical at the start of every PDP-11 instruction.

Floating point microcode uses all the registers in ASPLO to store some state of the machine during the execution of certain instructions. This is indicated by setting the General Registers Unequal bit, WHAMI<3>. Restoration always occurs at the end of the floating point instruction; the WHAMI bit is cleared following restoration.

2.2.2 ′ Reading From the Scratchpads

To move data from a particular scratchpad location to the ALU
input, the microword must enable the correct section onto
the bus, and it must specify the location within that section.

Three fields in the microword control address selection for the
A and B scratchpads: BSEL, ASEL, and RIF.

```
  43   42    41   40    39   38    37   36    35
┌──────────┬──────────┬──────────┬──────────┬────────────────┐
│   BEN    │   BSEL   │   AEN    │   ASEL   │      RIF       │
└──────────┴──────────┴──────────┴──────────┴────────────────┘
```
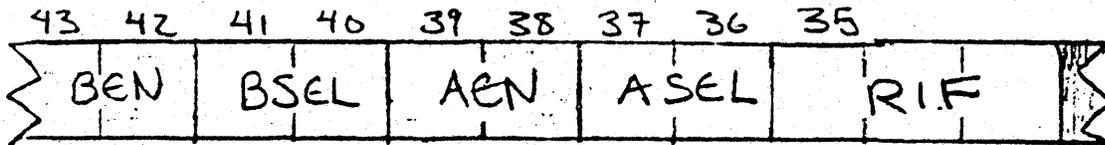
Figure 2-13: BSEL, ASEL, and RIF

BSEL and ASEL specify the way in which a location within the
scratchpad is addressed. Addressing can be either direct or
indirect; that is, an address in the scratchpad can actually
be specified, or a pointer to the source of the address can
be specified.

When the scratchpads are addressed directly, the Register
Immediate field (RIF), $\mu<35:33>$, is used in conjunction with
BSEL and ASEL to provide a full five-bit address specification.

The selection codes IMMED0 and IMMED1 specify the low-order
bit of the scratchpad address, and the RIF field specified
the three high-order bits. For timing reasons, RIF<2>, $\mu<35>$
is asserted low, and so that bit is inverted when used for
scratchpad addressing.

Figure 2-13 shows how the BEN, BSEL, and RIF fields work
together to specify an address in the BSP. The ASP works the
same way. Since there is only one RIF field direct addressing
places constraints on which registers can be concurrently
accessed by this method.

BEN   BSEL   RIF   LOCATION ADDRESSED

BEN/BSPLO

BSEL/IMMED0
RIF/0 → BSP [10]
RIF/1 → BSP [12]
RIF/2 → BSP [14]
RIF/3 → BSP [16]
RIF/4 → BSP [0]
RIF/5 → BSP [2]
RIF/6 → BSP [4]
RIF/7 → BSP [6]

BSEL/IMMED1
RIF/0 → BSP [11]
RIF/1 → BSP [13]
RIF/2 → BSP [15]
RIF/3 → BSP [17]
RIF/4 → BSP [1]
RIF/5 → BSP [3]
RIF/6 → BSP [5]
RIF/7 → BSP [7]

BEN/BSPHI

BSEL/IMMED0
RIF/0 → BSP [30]
RIF/1 → BSP [32]
RIF/2 → BSP [34]
RIF/3 → BSP [36]
RIF/4 → BSP [20]
RIF/5 → BSP [22]
RIF/6 → BSP [24]
RIF/7 → BSP [26]

BSEL/IMMED1
RIF/0 → BSP [31]
RIF/1 → BSP [33]
RIF/2 → BSP [35]
RIF/3 → BSP [37]
RIF/4 → BSP [21]
RIF/5 → BSP [23]
RIF/6 → BSP [25]
RIF/7 → BSP [27]

Figure 2-14: Direct Addressing of BSP

Alternatively, ASEL and BSEL can specify that fields in the current macro-level instruction are to provide the scratchpad address. The instruction's source register field, IR<8:6>, or the destination register field, IR<2:0>, may be specified. This allows more generality at the microcode level.

For example, if the PDP-11 instruction ADD R2, R3 is to be executed, there are two ways of addressing the operands:

```
A:      BEN/BSPLO, BSEL/IMMED0, RIF/5,     ; R[2] FROM BSP
           AEN/ASPLO, ASEL/IMMED1,            ; R[3] FROM ASP


B.      BEN/BSPLO, BSEL/SF,                ; R[2] FROM BSP
           AEN/ASPLO, ASEL/DF                 ; R[3] FROM ASP
```

You can see that the specifications in A are useful only when R2 and R3 are to be added, while those in B would work for any register-to-register add.

The encoding of the scratchpad addressing fields is shown in Table 2-4

Table 2-4: BSEL, ASEL Encoding

Enable field	Type of Addressing	Value Name	Field Value
AEN/ASPLO or AEN/ASPHI	RIF 0	IMMED0	0
	RIF 1	IMMED1	1
	R(DF)	DF	2
	R(SF)	SF	3
BEN/BSPLO or BEN/BSPHI	R(DF)	DF	0
	R(SF)	SF	1
	RIF 0	IMMED0	2
	RIF 1	IMMED1	3

2-22

Table 2-5 summarizes how the inversion of RIF<2> affects
direct register selection.

Table 2-5
RIF Summary

TOP 3 BITS OF REGISTER SELECTED	RIF CONTENTS
000	4
001	5
010	6
011	7
100	0
101	1
110	2
111	3

2.2.3 Writing Back to ASP and BSP

After clocking an ALU result into D at P2, you can write the
data into the A and B scratchpads during the same microcycle.
The primary purpose of the write-back is to update a
particular register, so address selection for write-back is
dependent upon the address chosen for reading.

This does not mean, however, that you have to write the same
location that was read. For example, consider the PDP-11
instruction ADD R2, R3 again. After execution, only the
contents of R3 should have changed. The implementation of this
instruction would contain the following specification:

    ALU/ADD, BEN/BSPLO, BSEL/SF,
        AEN/ASPLO, ASEL/DF, WHEN/P2, CLKD/YES

2-23

This indicates that R2 is to be read from the BSP, and R3
from the ASP. Recalling the rule mentioned earlier that
identical copies of the general registers must be maintained,
you can see that both BSP[3] and ASP[3] must be updated on
write-back. The address selection used to read from the ASP
should be used to write both scratchpads.

Therefore, while you can write the contents of D into BSP and
ASP simultaneously, the data goes into the same location in
both scratchpads. This mechanism ensures that both copies of
the destination register are updated correctly.

The Scratchpad Rewrite field, $\mu<19:14>$, is divided into a
number of subfields, as shown in figure 2-14.
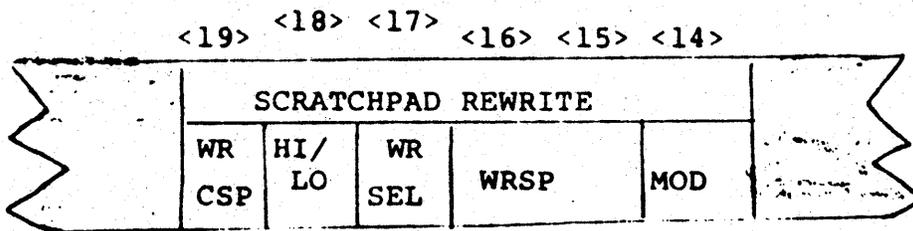
Figure 2-15: Scratchpad Rewrite Fields

MOD,$\mu<14>$, controls the interpretation of $\mu<18:15>$: it is
a steering bit as described in Section 1.5.1.3. The MOD
field must be 0 to write to ASP and BSP.

The Write Scratchpad (WRSP) field determines which scratchpad
is to be written: ASP, BSP, or both.

The Write Select (WR SEL) field specifies which address, as
specified in ASEL and BSEL, is to be used as the write-back
address.

HI/LO specifies which section of the scratchpad(s) is to be written. You can write-back to a different section than that specified by the Bus Enable fields.

Write CSP (WR CSP) controls writing of the C scratchpad.

The encoding of these fields is shown in Table 2-6.

Table 2-6
Scratchpad Rewrite Fields

FIELD	ACTION	MNEMONIC	FIELD VALUE
MOD, μ<14>	μ<18:15> controls scratchpad rewrite	CLKSP	0
WRSP, μ<16:15>	Write ASP only	WR A	1
	Write BSP only	WR B	2
	Write both ASP and BSP	WR A AND B	3
	Do ʹ⸱⸱⸱ ˁ⸱⸱⸱ʹ SPAD	ʹ C	3
WRSEL, μ<17>	Use ASP address	A ADDRS	0
	Use BSP address on rewrite	B ADDRS	1
HI/LO, μ<18>	Write LO section of SPAD	LO	0
	Write HI section of SPAD	HI	1

Now we can add some more specifications to our microinstruction for ADD R2, R3:

```
ADD:    ALU/ADD, BEN/BSPLO, BSEL/SF
            AEN/ASPLO, ASEL/DF, WHEN/P2, CLKD/YES, HILO/LO,
                WRSEL/A ADDRS,  WRSP/WR A and B,
                    MOD/CLKSP
```

Scratchpad rewrite always occurs at P3, so the D register must be clocked at P2 if you wish to write back to the scratchpads during the same microcycle.

2.3   THE C SCRATCHPAD

The third source on BUS BIN is the C scratchpad (CSP), which is 16 registers deep. It is the means by which data from the outside world (i.e., main memory or other sections of the processor) is introduced into the datapath.

The CSP is also used to store constants and error log information.

2.3.1   The Base Constants

Three locations in the CSP are permanently reserved for the base constants of the machine: zero, one, and two. CSP[17] contains the value one; CSP[16] contains the value zero; and CSP[14] contains the value two. These locations MUST NOT be changed.

By convention, CSP[15] stores data from the outside world. Since this is usually data from memory, CSP [15] is called the Memory Data register, or MD.

These four locations in the CSP, CSP [14:17], have a special addressing mechanism, and a special BEN field value may be used to access them.

LOCATION	NAME(S)	CONTENTS
CSP(0) LOG.JAM	CNST4	000004
CSP(1) LOG.SERVICE	CNST8	000010
CSP(2) LOG.PBA	RESRIGHT	020000
CSP(3) LOG.CUA	EXPMASK	077600
CSP(4) LOG.FLAG/INTR	RESLEFTD(C)	050000
CSP(5) LOG.WHAMI	RESLEFTGD	054000
CSP(6) LOG.CACHEDATA	EMITCON	
CSP(7) LOG.TAG/CPU	RESRIGHTGD	024000
CSP(10) CNSL.CNST100000	HIBYTEMASK	177400
CSP(11) CNSL.CNST177770	SEXPMASK	177600
CSP(12) CNSL.CNST30000	SIGNBIT	100000
	CNST100000	
CSP(13)	CNST200 HIDDENBIT EXPONE SETDMASK	000200
CSP(14)	# 2	000002
CSP(15)	MD	******
CSP(16)	# 0	000000
CSP(17)	# 1	000001

Figure 2-16   CSP Layout

2-27

When the BEN field of the microword contains the value 3, the
BSEL field selects one of the four special locations in the
CSP. The encoding is as follows:

> When BEN/3, then:
> | | | |
> |---|---|---|
> | BSEL/0 | selects CSP[17] | (1) |
> | BSEL/1 | selects CSP[16] | (0) |
> | BSEL/2 | selects CSP[15] | (MD) |
> | BSEL/3 | selects CSP[14] | (2) |

2.3.2   Other Locations in the CSP

You may use CSP[0:13] to store data, subject to certain
restrictions. These locations usually hold constants, such
as a mask for isolating the exponent field in a floating point
number, which are needed by various segments of the base machine
code.

The 11/60's Emit facility, described in Chapter 4, enables you
to store arbitrary constants in the CSP, after setting a flag
in another section of the processor.

When the value of the BEN field is not equal to 3, the CSPADR
field provides the CSP address, in much the same way as RIF
provides an address in the ASP and BSP.

CSPADR, $\mu<23:20>$, holds the complement of an address in the
CSP. That is, the bits in the CSPADR field are complemented
before they select a location, as shown in Table 2-7.
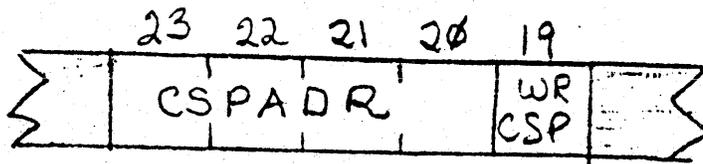
Figure 2-17:   CSPADR, WRCSP Fields

Table 2-7

CSP ADDRESSING

CSPADR Field	Bit Patterns	Complemented Pattern	CSP Location Selected
0	0000	1111	17
1	0001	1110	16
2	0010	1101	15
3	0011	1100	14
4	0100	1011	13
5	0101	1010	12
6	0110	1001	11
7	0111	1000	10
10	1000	0111	7
11	1001	0110	6
12	1010	0101	5
13	1011	0100	4
14	1100	0011	3
15	1101	0010	2
16	1110	0001	1
17	1111	0000	0

Thus to read from the CSP, use BEN codes BASCON or CSP, and specify the address with BSEL or CSPADR, respectively.

2.3.3   Writing to the CSP

The CSP's input data comes from the DMUX, which accepts data from the Cache and from main memory and other sections of the processor. You do not have to control this multiplexer: it will automatically select the correct source.

The WRCSP field, $\mu<19>$, controls writing to the CSP. If the WRCSP field contains a 1, the output of the DMUX will be written to a location in the CSP at P3. If WRCSP contains a 0, no data will be written.

If the microinstruction contains the specifications

BEN/BASCON, BSEL/MD, WRCSP/YES

then the data will be written into CSP[15], MD. (Remember that you must not over-write any other base constant.) Otherwise, the WRCSP/YES specification will cause data to be written into the location specified by the complement of the CSPADR field.
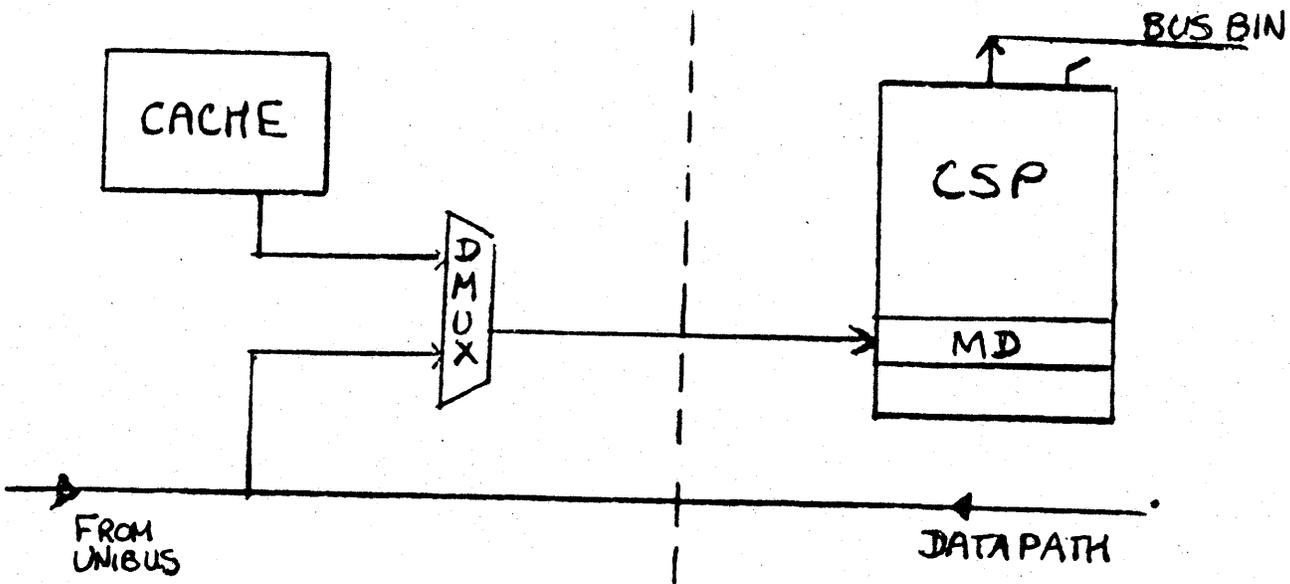
Figure 2-18: Writing Data to CSP

If you write data to any location in the CSP other than MD, you must set a flag in the Processor Control Section. This flag, CSP CONSTANTS INVALID, indicates that the constants needed by the floating point microcode are not available. The mechanism for setting the flag is described in Section 4.2.4.4.

The constants used by the floating point microcode are shown in Figure 2-16; if the CSP CONSTANTS INVALID flag is not set, you can use these constants in your routines.

Note that the two methods of CSP addressing are mutually exclusive. You cannot read one CSP location and write to another in the same microinstruction.

2.4   THE XMUX AND THE SHIFT REGISTER

The XMUX is a two-to-one multiplexer which, when selected by AEN, puts its output onto BUS AIN. One of the XMUX sources is the 16-bit Shift Register, described in Section 2.4.1.

When AEN = 0, the XMUX field of the microword, $\mu<36>$, controls XMUX selection. Note that this field overlaps the ASEL field. Be careful not to specify WR SEL/A ADRS if writing back to the scratchpads after using the XMUX as the AIN source.
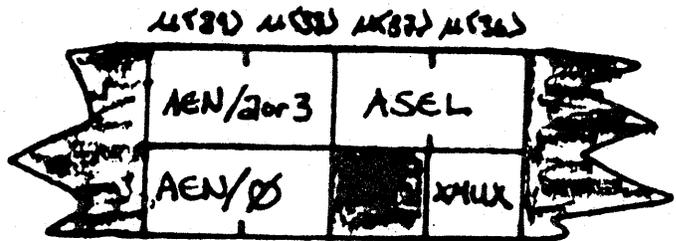
Figure 2-19: XMUX, AEN, and ASEL Fields

When the value of the XMUX field is 0, the output of the SR goes onto BUS AIN. When the XMUX field contains the value 1, a word of the form shown in Figure 2-20 is put on BUS AIN.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D(C)	0	0	0	0	0	0	0	0	← SR<6:0> →						
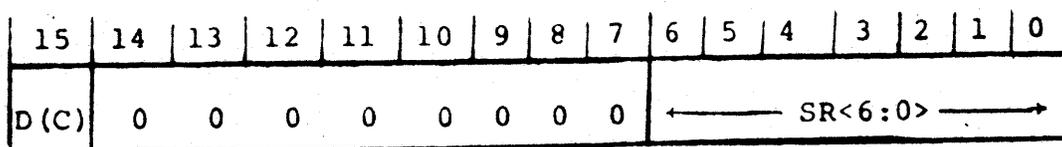
Figure 2-20: S1 XMUX Input

2.4.1   The Shift Register

The Shift Register is a 16-bit bidirectional shift register.
It has four distinct modes of operation:

> Parallel load from ALU output (default)
> Shift right one bit per microcycle
> Shift left one bit per microcycle
> Do nothing

Mode control for the SR is provided by the Residual Control
register, which is described in Section 2.9.

The SR, when in parallel load mode, is loaded with the output
of the ALU.

Regardless of the operating mode of the SR, its clocking is
controlled by the WHEN, $\mu<29>$, and CLKSR, $\mu<27>$, fields of the
microword. The SR is clocked if the CLKSR field contains a 1.
Clocking occurs at P2 is WHEN equals 0, and at P3 if WHEN
equals 1. If both D and SR are clocked in the same microcycle,
they are clocked at the same time, and receive the same data.

A 16-way branch can be performed on the basis of SR<3:0>. This
facility, called the CASE branch, is described in Section 3.6.2.
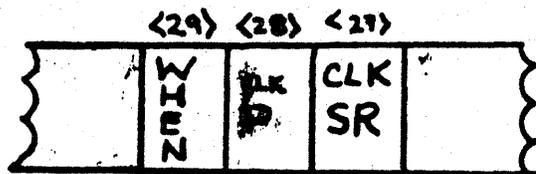
Figure 2-21   WHEN, CLKSR Fields of µword

2.5   THE SHIFT TREE

The final A-side source is the Shift Tree, or barrel shifter.
This is the major element of the 11/60's field isolation unit.
The Shift Tree performs various operations on data from the
D register; these operations include:

        Left Shift 1 bit per microcycle
        Right Shift 1,2,3, ... 14 bits per microcyle
        Sign Extend
        Byte Swap

Unlike the Shift Register, the Shift Tree is a combinational
logic element and thus does not hold its output across micro-
cycles. It is designed so that data clocked into D in a
previous microcycle can be modified in the Shift Tree, operated
upon by the ALU, and the result stored -- all during one micro-
cycle.

The data to be manipulated must be stored in D by P2 of the
microcycle preceding the Shift Tree operation. The data can
then be clocked into D and stored in the scratchpads, as
illustrated in Figure 2-24.

   Figure 2-24:   D to D to Scratchpad in one Microcycle

←——— μcycle α ———————→	←—μcycle α + 1———————→
P2: D ← DATA	D
	↓
	TREE
	↓
	BUS AIN
	↓
	ALU ← BUS BIN DATA
	↓
	P2: D ← DATA
	↓
	P3: SPAD ← DATA
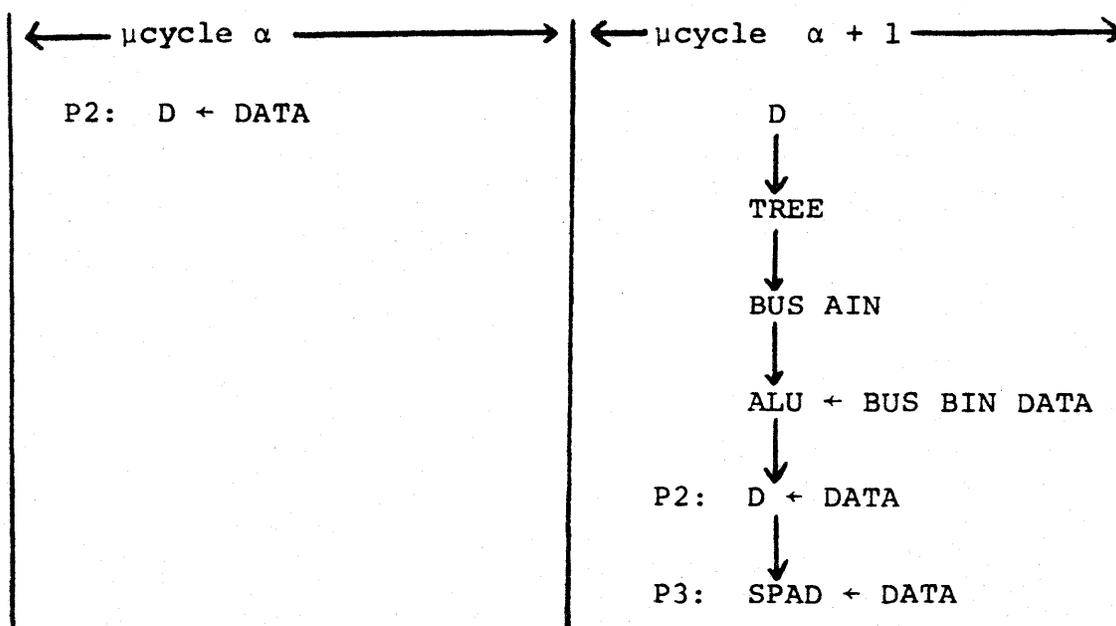
Although you will use macros to control the Shift Tree, you must look closely at the hardware involved.

There are three levels of multiplexers, interconnected to effect shifting, in the Shift Tree. This layout is shown in Figure 2-26. The contents of D are input to the AMUX, the output of the AMUX is the input for the BMUX; the BMUX output goes into the CMUX, and the CMUX output goes onto BUS AIN.

To perform a particular operation, you must specify a multiplexer selection for each stage of the Shift Tree. Thus, to shift the D output right by six, specify:

AMUX/DIRECT, BMUX/RIGHT-FOUR, CMUX/RIGHT-TWO.

To specify a right shift of seven:

AMUX/RIGHT-EIGHT, BMUX/DIRECT, CMUX/LEFT-ONE

Note that the Shift Tree is not a circular shifter. That is, bits shifted off one end are not shifted into the other end of the word.

The three fields in the microword that control the selection of the stages of the Shift Tree are: AMUX, μ<22:20>; BMUX, μ<23>; and CMUX, μ<37:36>.
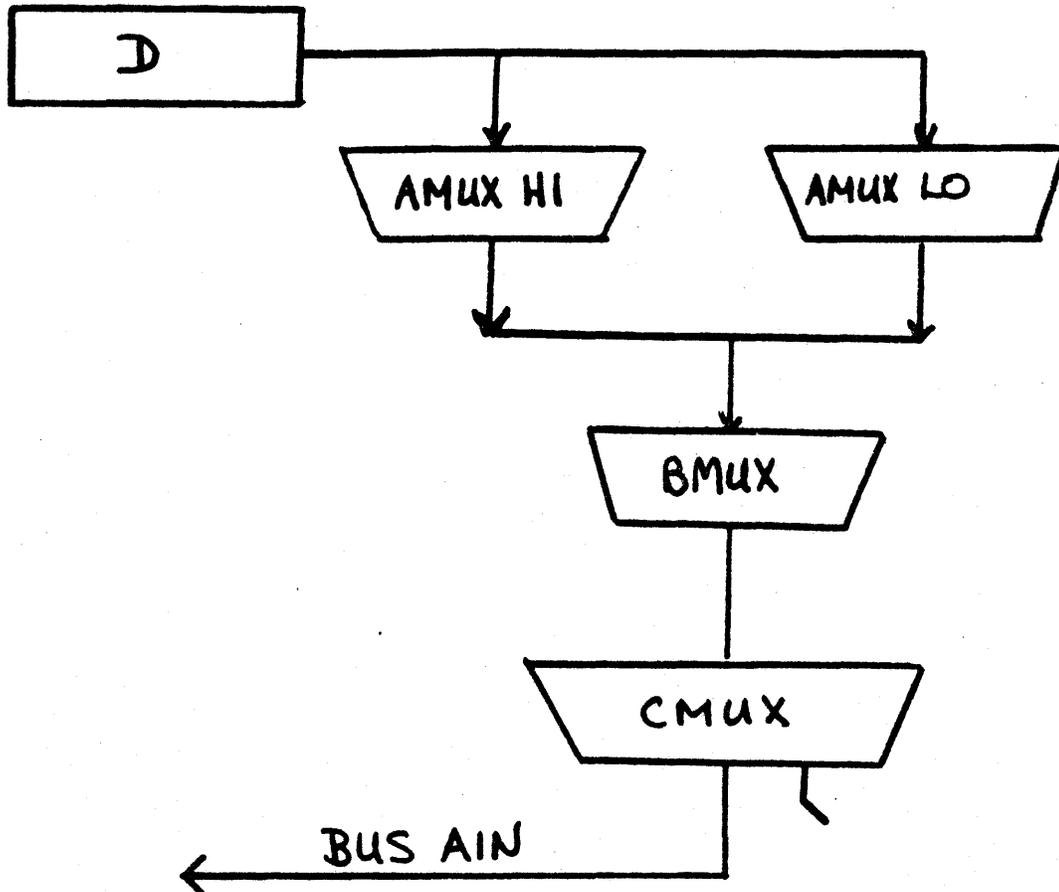
Figure 2-26   Simplified Shift Tree
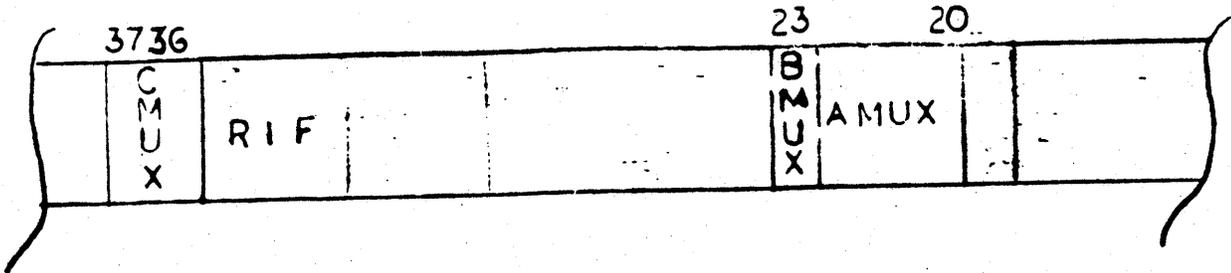
Figure 2-27   Shift Tree Control Fields

Again, you will notice that these fields overlap fields
previously discussed. Because the AMUX and BMUX fields occupy
the same bits as CSPADRS, CSP access during Shift Tree oper-
ations is contrained to those locations which can be addressed
with BSEL: the base constants.

The encoding of the Shift Tree control fields is shown in
Table 2-8.   The detailed diagram of the Shift Tree (see Figure
2-28) should clarify the entries in Table 2-8. The bits in
the microword fields are the source of selection signals for
the three levels of multiplexer. Thus CMUX<0> is the source
of the signal CMUX S∅, and so forth.

Figure 2-28 also shows how the choice of signals going into
each input data port effects the shifting actions of the
Shift Tree.
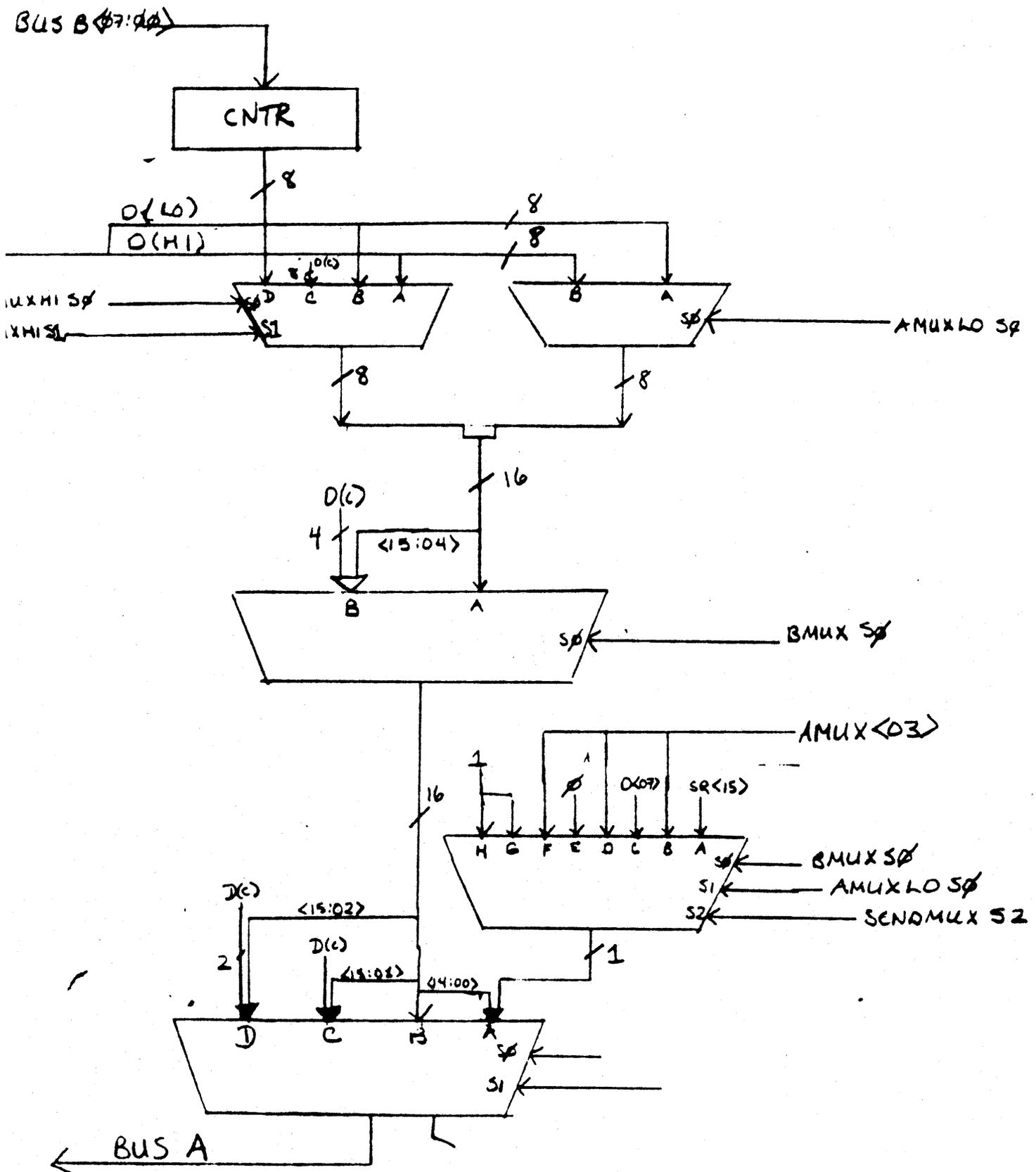
Figure 2-28: Details of the Shift Tree

2.5.1 AMUX and CNTR

AMUXHI provides the high byte of the AMUX output; AMUXLO
provides the low eight bits. The high and low bytes of the
D output are separate inputs into each AMUX. This allows
duplicating either byte, swapping bytes, and shifting eight
bits to the right. (The right shift consists of selecting
D<15:08> as the low byte of the AMUX output, and filling in
the high byte with D(C).)

The Counter (CNTR) register, at the top of Figure 2-28, is
an iteration counter. It is not part of the Shift Tree.
However, the AMUX can introduce the contents of CNTR into the
datapath. It is described in Section 2.7.

2.5.2 The BMUX

The BMUX can either pass the output of the AMUX without change,
or it can shift the AMUX output right by four, filling in
the high bits with D(C).

2.5.3 The CMUX and SENDMUX

The CMUX can perform a right shift by one or two; pass the
BMUX output without change; or shift left by one bit. The
Shift End Multiplexer, SENDMUX, provides the low-order bit
when the CMUX is shifting left. D(C) fills in high-order bits
when shifting right.

Table 2-8
Shift Tree Control

AMUX FIELD (μ<22:20>) ENCODING		
Function (Output)	Mnemonic	Field Value
D unchanged	DIRECT	Ø
D<7:0>in both bytes	DLO#DLO	1
D(C) fills high byte, D<7:0> in low byte	SIGNEXT	2
Contents of Counter in high byte, D<7:0> in low	COUNTER	3
D<15:08> in both bytes	DHI#DHI	4
Swap bytes	SWAB	5
D(C) fills high byte, D<15:08> in low byte	RIGHT-8	6
Counter in high byte, D<15:08>in low byte	COUNTER#DHI	7

BMUX FIELD (μ<23>) ENCODING		
Output of AMUX unchanged	DIRECT	Ø
Shift output of AMUX right four, D(C) fills high bits	RIGHT-4	1

CMUX FIELD (μ<37:36>) ENCODING		
Output of BMUX left one with SENDMUX into low bit	LEFT-1	Ø
Output of BMUX unchanged	DIRECT	1
Output of BMUX right one with D(C) into high bit	RIGHT-1	2
Output of BMUX right two with D(C) into high bits	RIGHT-2	3

You cannot control the SENDMUX directly from the microword because the source of the bit shifted into the zero bit of the CMUX output usually depends on what was done in the higher stages of the Shift Tree. To illustrate how this works, look again at the example of a right shift by seven.

The final CMUX output should be a word with D(C) in the high seven bits, and D<15:07> in CMUX<8:0>. In the example:

AMUX/RIGHT-8	(8*D(C) # DHI)
BMUX/DIRECT	(No Change)
CMUX/LEFT-1	(Left One)

But you can see from Figure 2-28 that D<07> will not go through the BMUX to the CMUX; in effect, it falls off the end of the AMUX. The SENDMUX "catches" this bit. When AMUX<02>, µ<22>, is set - making D<15:08> the output of AMUXLO - and no shift is indicated for the BMUX, the SENDMUX output is D<07>. This becomes the low bit of the CMUX output, and the shift is completed correctly.

Similarly, if a shift of 11 right (AMUX/RIGHT-8, BMUX/RIGHT-4, CMUX/LEFT-1) or 3 right (AMUX/DIRECT, BMUX/RIGHT-4, CMUX/LEFT-1) is specified, bit 3 of the AMUX output falls off the end of the BMUX. In both cases, the SENDMUX correctly feeds this bit into the CMUX.

These effects are possible because the S∅ and S1 selection ports of the SENDMUX are controlled by BMUXS∅ and AMUXLOS∅, respectively. The thrid selection port, S2, is controlled from the RES register (see Section 2.9). Table 2-9 is the SENDMUX truth table.

2.6 SHIFTING WITH THE SHIFT REGISTER

The shifting capabilities of the Shift Tree and the Shift
Register are somewhat interdependent, thus, before presenting
more examples of Shift Tree operations, the following sections
describe the Shift Register's shifting modes.

2.6.1   The SR GUARD

There is a 4-bit extension to the Shift Register called the
SR Guard (GUARD), for use by the microcode floating point.
The GUARD is the same type of register as the SR, and has
the same four operating modes. It is clocked at the same
time as the SR when it is enabled from the RES register.

When the RES register specifies parallel load for the SR,
the GUARD is loaded with zeroes.

Conditional braches can be made on the contents of GUARD <3:2>;
see Section 3.3.
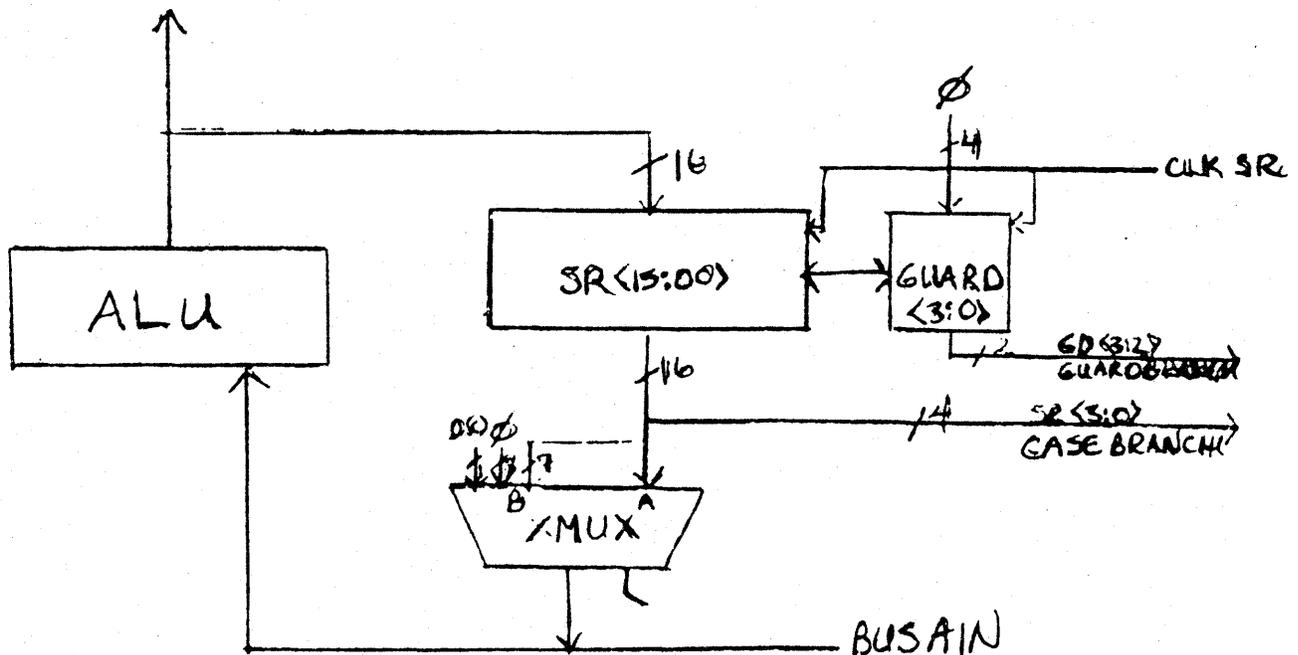
Figure 2-22: SR, GUARD Registers

2.6.2   Right Shift

When a right shift is indicated, the previously loaded 16-bit
word in the SR is shifted right one bit position. BMUX<00>,
from the Shift Tree, fills SR<15>. If the Guard register
is enabled, SR<00> fills GD<03>. Bits shifted out of GD<00>
are lost.

The wiring of the SR and Guard registers for a right shift is
shown in Figure 2-22.
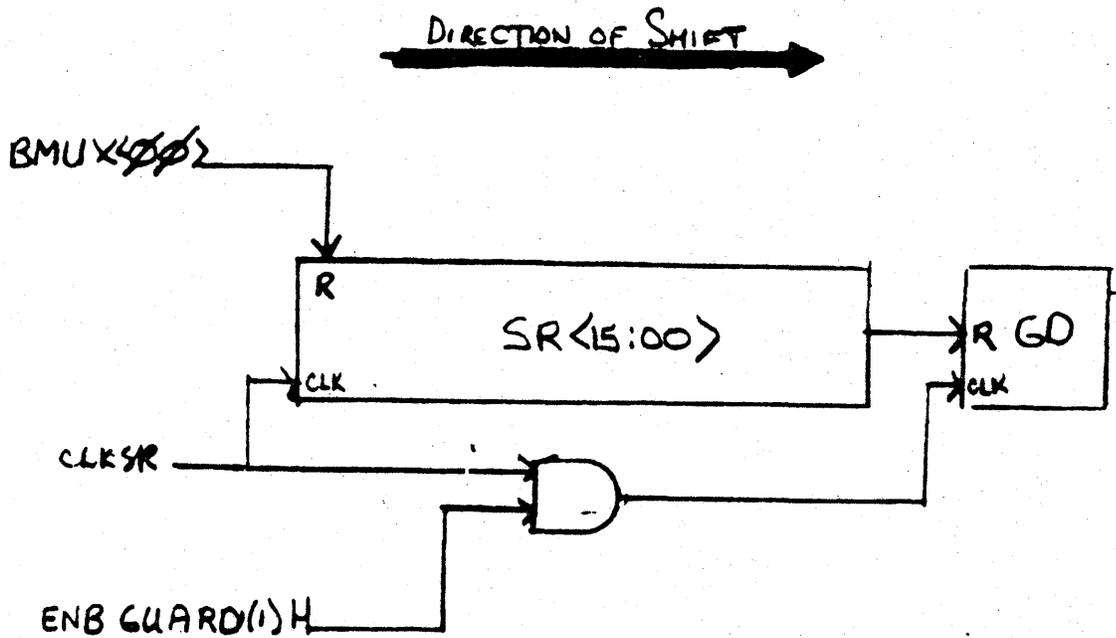
Figure 2-22:   Right Shift of SR

2.6.3   Left Shift

When a left shift is indicated, either GD<03> or D(C) can be
shifted into SR<00>. SR<15> is shifted into SENDMUX<0>,
where it can be directed into CMUX<00> (see Section 2.5).

The high-order Guard bit is shifted into the SR only if the
Guard register is enabled from the RES register. Figure
2-23 illustrates the wiring of the SR and Guard registers
for a left shift.
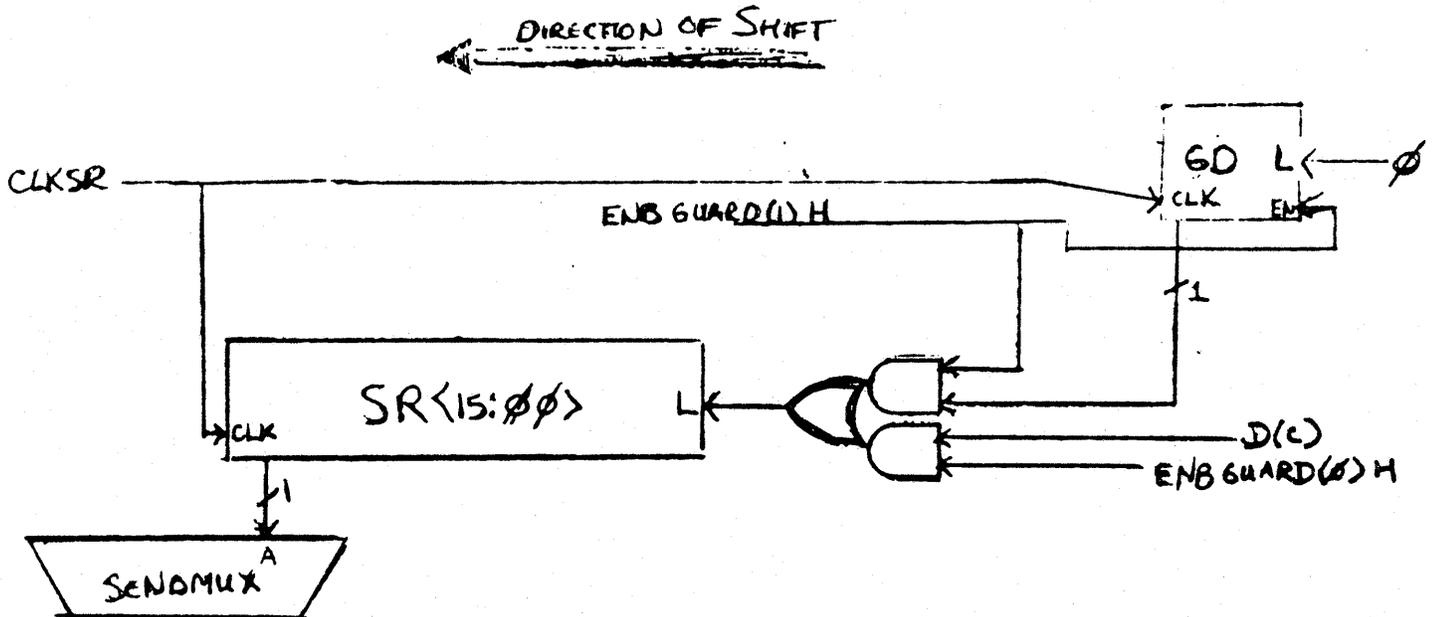
Figure 2-23: Left Shift of SR

The particular routing of the shift outputs and inputs for
the SR are designed to allow the SR and D to function as a
32-bit shift register. Examples are shown in Section 2.7.

2.7   SHIFT EXAMPLES

2.7.1   Multiple-Word Shifts

When AMUXLO selects the low byte of the D data, and the BMUX
passes its input without alteration, SR<15> can be directed
into the CMUX from the SENDMUX. This enables the Shift Tree
to act as the high-order part of a 32-bit shift register.
While the low-order word is shifted one bit to the left in
the SR, the high-order word, previously stored in D can be
shifted in the Shift Tree and returned to the D register.
This action is illustrated in Figure 2-29.

<table>
<tr><td>In previous cycles</td><td>1 microcycle</td></tr>
<tr><td>RES set up for
left shift</td><td>AEN/CMUX, ALU/SELECT A,
AMUX/DIRECT, BMUX/DIRECT,</td></tr>
<tr><td>High-order word in
D register</td><td>CMUX/LEFT-1, CLKD/YES,</td></tr>
<tr><td>Low-order word in
Shift Register</td><td>CLKSR/YES, WHEN/P2</td></tr>
</table>

Figure 2-29: Left Shift on 32 Bits of Data

A right shift on 32 bits of data can be accomplished in a
similar fashion. Recall that when the SR is shifted right,
the low bit of the BMUX output is shifted into SR<15>. So
by setting up the data and the SR mode control for a right
shift, and then specifying:

        ALU/SELECT A, AEN/CMUX, AMUX/DIRECT, BMUX/DIRECT,
            CMUX/RIGHT-1, CLKD/YES, CLKSR/YES, WHEN/P2

you will shift D<00> into SR<15>. Figure 2-30 illustrates
the result if the Guard register was enabled.

A: ddddddddddddddd
                D

                          sssssssssssssss   0000
                                SR   GD

B: D(C)ddddddddddddddd
                D

                          dssssssssssssss   s000
                                SR   GD

Figure 2-30:   Right Shift on 32 Bits of Data

Table 2-9

SENDMUX TRUTH TABLE

S2	S1	SØ	CMUX<00> input
0	0	0	SR<15>
0	0	1	AMUX<03>
0	1	0	D<07>
0	1	1	AMUX<03>
1	0	0	0
1	0	1	AMUX<03>
1	1	0	undefined
1	1	1	undefined

2.6   Shift Examples

This section contains simple microcode equivalents for a
number of PDP-11 shift instructions. A symbolic description
of the actions of each microinstruction and the field value
specifications are shown.

2.6.2   ASL R0

The execution of ASL R0 would take at least two microcycles.
In the first,

        P2: D ← ASP [0]

and in the second,

        P2: D ← D LEFT ONE

        P3: ASP [0] ← D

        P3: BSP [0] ← D

The field specifications would be as follows:

        INSTR1:

                ALU/SELECT A, AEN/ASPLO, ASEL/IMMED0
                RIF/4, CLKD/YES, WHEN/P2

        INSTR2:   ALU/SELECT A, AEN/CMUX, BEN/BSPLO,
                    BSEL/IMMED0, RIF/4, AMUX/DIRECT,
                    BMUX/DIRECT, CMUX/LEFT ONE, CLKD/YES,
                    WHEN/P2, MOD/CLKSP, HILO/LO, WRSEL/B ADDR,
                    WRSP/A AND B

Notice that in the second microinstruction, a BEN and a BSEL
value were specified, even though the ALU function was only
to pass the data on BUS AIN to D. The BSP address selection
is used to set up the correct write-back address. The
SENDMUX would have to be set up from RES if you wanted a ∅
shifted into the low-order bit.

2.6.3   ASR R1

Symbolic specification:

            INSTR1:   P3: D ← R1
                        P3: D(C) ← ALU<15>

            INSTR2:   P2: D ← D RIGHT ONE
                        P3: ASP[1] ← D
                        P3: BSP[1] ← D

Field value specifications:

        INSTR1:

                    ALU/SELECT A, AEN/ASPLO, ASEL/IMMED1
                        RIF/4, COUT/ALU15, CLKD/YES,
                        WHEN/P2

        INSTR2:

                    ALU/SELECT A, BEN/BSPLO, BSEL/IMMED1,
                        AEN/CMUX, AMUX/DIRECT, BMUX/DIRECT,
                        CMUX/RIGHT ONE, MOD/CLKSP, HILO/LO,
                        WRSEL/B ADDRS, WRSP/A AND B, CLKD/YES,
                        WHEN/P2

2.6.4   ASH #-11, R∅

In this example, the indirect addressing of the B and A scratch-
pads is exploited to make the example more general.

Symbolic specification:

            INSTR1: P2: D ← R(SF)
                     P2: D(C) ← ALU<15>

            INSTR2: P2: D ← D RIGHT 11
                     P3: R[SF] ← D

Field value specifications:

        INSTR1:   ALU/SELECT A, AEN/ASPLO, ASEL/SF,
                     CLKD/YES, WHEN/P2, COUT/ALU15

        INSTR2:   ALU/SELECT A, BEN/BSPLO, BSEL/SF,
                     AEN/CMUX, CLKD/YES, WHEN/P2,
                     AMUX/RIGHT EIGHT, BMUX/RIGHT FOUR,
                     CMUX/LEFT ONE, MOD/CLKSP, HILO/LO,
                     WRSEL/B ADDRS, WRSP/A AND B

2.7   The Counter Register

The Counter Register (CNTR) is an eight-bit counter. It can
be used to control repeated loops through the datapath. Its
loading is controlled by MOD, μ<14>, and CLK CNTR, μ<16>. If
both MOD and CLK CNTR contain the value 1, the CNTR is loaded
from BUS BUS<07:00>. (If MOD equals 0, μ<16> is interpreted
as part of the WRSP field.)

The COUNTER counts up, not down, so the value loaded from
BUS BIN must be the complement of the actual count. For
timing reasons, it must be loaded with the 2's complement of
the count.

Incrementing and clearing the COUNTER are controlled by Active
Branches, which are described in Section 3.6.

2.8   THE BA REGISTER

Because addresses are relocated through the KT unit, the physical
addressing of main memory is transparent to the 11/60 micro-
programmer. To access a Unibus location, you will specify its
virtual address. (only the console microcode uses physical
addresses)

The Bus Address (BA) register holds the address for data
coming from or going to a Unibus location. Thus, when data
from memory is to be moved into MD (by a DATI), you load
BA with the virtual address of the location to be read.
Similarly, when data in D is to be written to main memory (by
a DATO), specify the address of the location with BA.

The virtual address is loaded into BA from BUS AIN, as shown
in Figure 2-31. The two high-order bits can, in special cases,
be loaded from BUS BIN<01:00>; normally they are set to 0 by
Bus Control logic. You do not have to worry about the data
on BUS BIN affecting the Bus Address.

The output of the BA register goes to the Memory Management
unit (KT), where it is mapped to a physical address. This
physical bus address is then used by both the Cache and the
Unibus.

If, on a DATI, the location specified by the BA and relocated
by the KT unit is available in the Cache, no Unibus access is
made. If a Unibus DATI cycle is performed, however, the
Cache is updated when the data is brought in from main
memory. On a DATO cycle, main memory and Cache are both
updated.
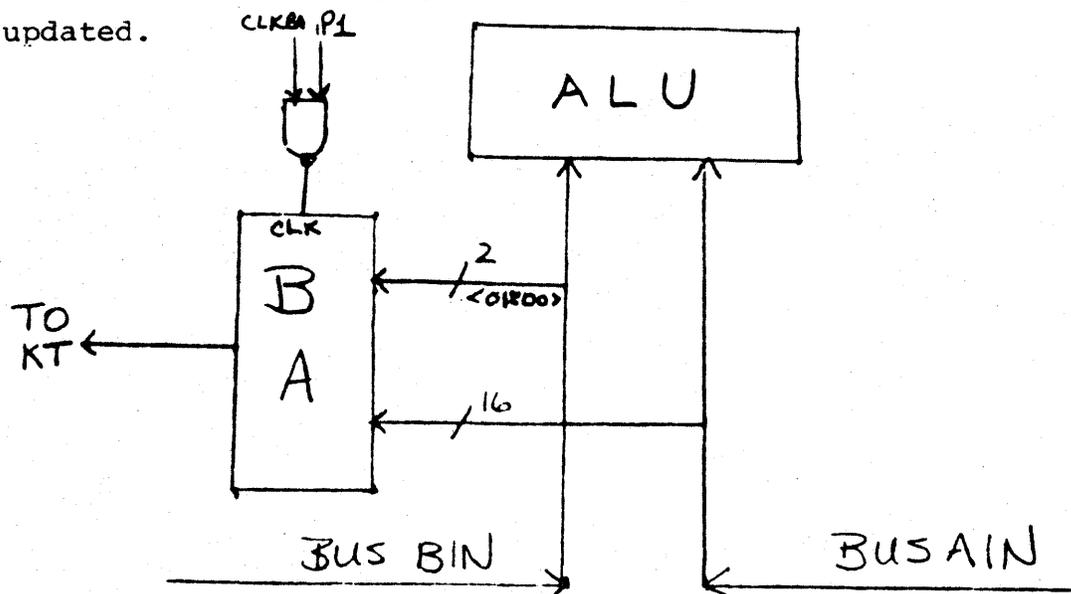
Figure 2-31: The BA Register

2-49

BA loading is controlled by the CLKBA field of the microword, μ<26>. When CLKBA contain the value 1, the BA register is loaded at P1. (The value of the WHEN field has no effect upon the clocking of BA.) The BA is clocked earlier than other registers to allow for cache cycle time. The requested data is available at the CSP input at P3 of the <u>following</u> microcycle.
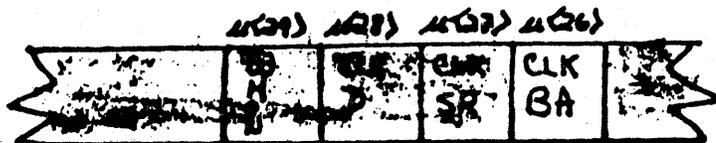
Figure 2-32: CLKBA Field of μword

2.9   THE RESIDUAL CONTROL CONCEPT

Two of the primary design goals for a microprogrammable machine are flexible control of the elements of the data-path and efficient use of the control store. These goals are occasionally at odds with one another, and various techniques have been developed to minimize the trade-off penalties.

One of these techniques is the use of distributed control, in which the central control store does not control all of the functional untis of the processor. Residual control (which is essentially a special case of distributed control) is used in the 11/60 to avoid widening the microword.

2.9.1   Set-up Registers

Much of the control information for a microprocessor is relatively static; that is, it is not changed every micro-cycle. This static information can be filtered out of the

microword and placed in special registers, called set-up
registers or stats. These set-up registers can then be used
in association with fields in the microinstruction to fully
define the control for a particular resource. This situation
is illustrated in Figure 2-33.
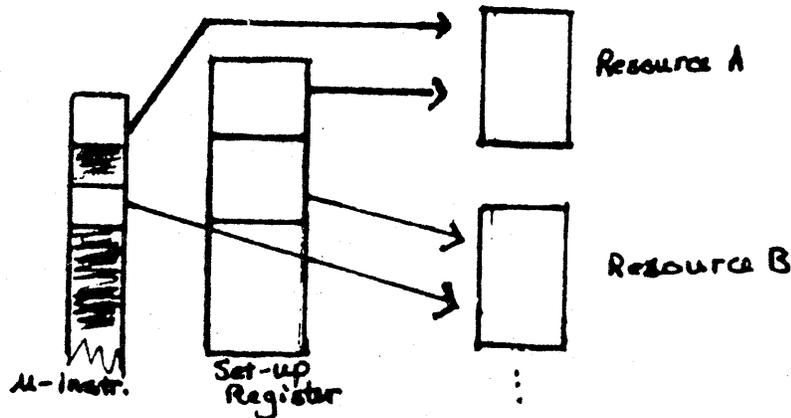
Figure 2-33: Set-up register

2.9.2   The RES Register

The Residual Control register, RES, controls the operating mode
of the SR and GD registers; selects the shift left input of
SR<00>; sets up SENMUX S2 for the end-shifted bit for CMUX in
the Shift Tree; and controls clocking of the Guard Register.

RES is loaded from BUS BIN<14:11> at P2 when MOD, μ<14>, and
CLKRES, μ<18>, are both equal to 1. Inputs and corresponding
outputs of the RES register are shown in Figure 2-34.
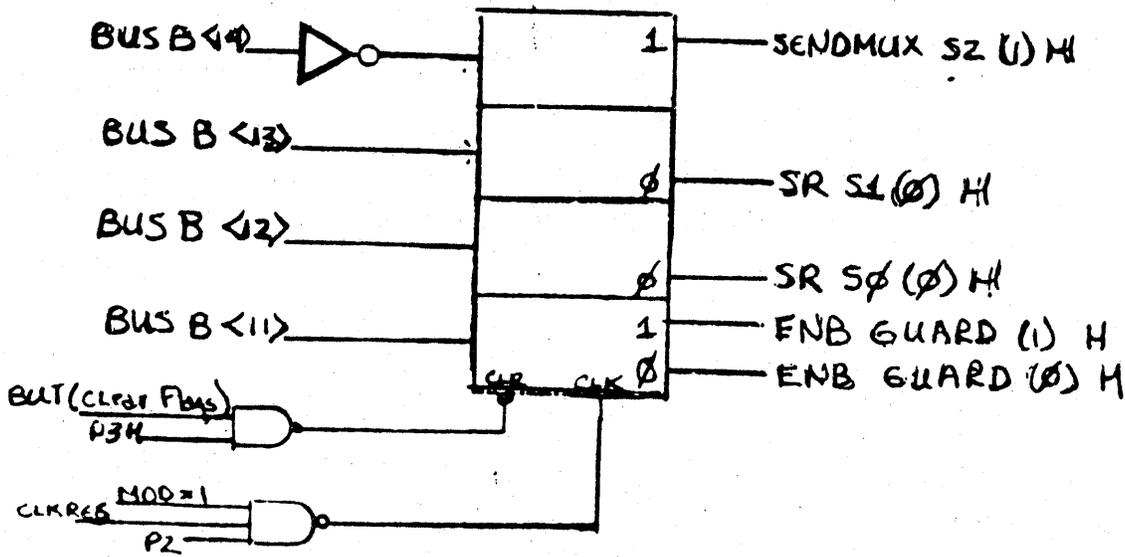
Figure 2-34: The RES register

2.9.2.1   SENDMUX S2 (1) H -- BUS BIN<14> is inverted before
it is stored into RES. The corresponding output signal is
SENDMUX S2 (1) H, which controls the S2 selection port of
the Shift End multiplexer.

If the SENDMUX S2 (1) signal coming from the RES register is
low, and both the AMUX and BMUX pass their input data unmodified,
then SR<15> becomes CMUX<00>. SENDMUX S2 will be low only if it
is loaded with B<14> equal to one.

2.9.2.2   SR Mode Control -- Mode control for the Shift
Register is provided by RES outputs SR S1 (Ø) and SR SØ (Ø).
These bits are the inverse of the values loaded from BUS BIN
<13:12>. Table 2-10 shows the truth table for the SR.

Table 2-10: SR Truth Table

S1	S∅	SR Function	BUS BIN<13:12> Values
0	0	Do nothing	11
0	1	Right Shift	10
1	0	Left Shift	01
1	1	Load	∅∅

The default mode, that is, the SR mode when RES is cleared, is to parallel load.

2.9.2.3   Guard Enable

The Guard register is clocked only if ENB Guard (1) is high. The BUS BIN<11> input to the RES register provides two output signals: ENB GUARD (1) and ENB GUARD(0). The Guard register is clocked only if ENB GUARD (1) is high; -that is, if BUS BIN<11> is equal to one when RES is loaded.

When ENB GUARD (∅) is high, the GUARD register is not clocked. Moreover, during a left shift, D(C) is shifted into the low bit of the SR.

2.9.2.4   Constants for Loading the RES Register -- The simplest way to load the RES register is to store a constant in the CSP, and direct it onto BUS BIN when you want to load RES. Table 2-11 shows the constant with which to load the RES register for particular functions.

Table 2-11: Constants for Loading RES

Function	Constant	BUS BIN Bits 14 13 12 11			
Shift SR right; GUARD register not enabled	020000	0	1	0	0
Shift SR left; D(C) into SR<00>, SR <15> into CMUX<0> if AMUX and BMUX go direct; Guard not enabled	050000	1	0	1	0
Shift SR left; GUARD<3> into AR<0>; SR<15> into CMUX<00> if AMUX and BMUX pass their input data unmodified; Guard enabled.	054000	1	0	1	1
Shift SR right; SR<0> into GUARD<3>; Guard Enabled.	024000	0	1	0	1
Direct AMUX<03> into CMUX<00> SR and GUARD not enabled (note that because of inversion of BUS BIN<14>, this is not the same as clearing RES).	000000	0	0	0	0

Notice that both RES and CNTR can be loaded from BUS BIN at
the same time,   because

$$CNTR \leftarrow BUS\ BIN<7:0>$$
$$RES \leftarrow BUS\ BIN<14:11>$$

For example, suppose you want to do 16 right shifts, as in a
multiply loop. The constant 020360 from the CSP would set up
the CNTR for a count of 16, and RES for a right shift in the
SR, as shown in the Figure on the following page

```
BUS BIN bits    15  14  13  12  11  10  9   8   7   6   5   4   3   2   1   0

020360  =       0   0.  1   0   0   0   0   0   1   1   1   1   0   0   0   0

don't care─────────┘   ↑   ↑   ↑   ↑           ↑               ↑

don't care ────────────┘

set SR S1 to.0 ────────────┘

set SR S0 to 1 ────────────────┘

GD not enabled─────────────────────┘

don't care─────────────────────────────────────────┘

2's complement of 16₁₀ - 20₈ ───────────────────────────────────┘
```
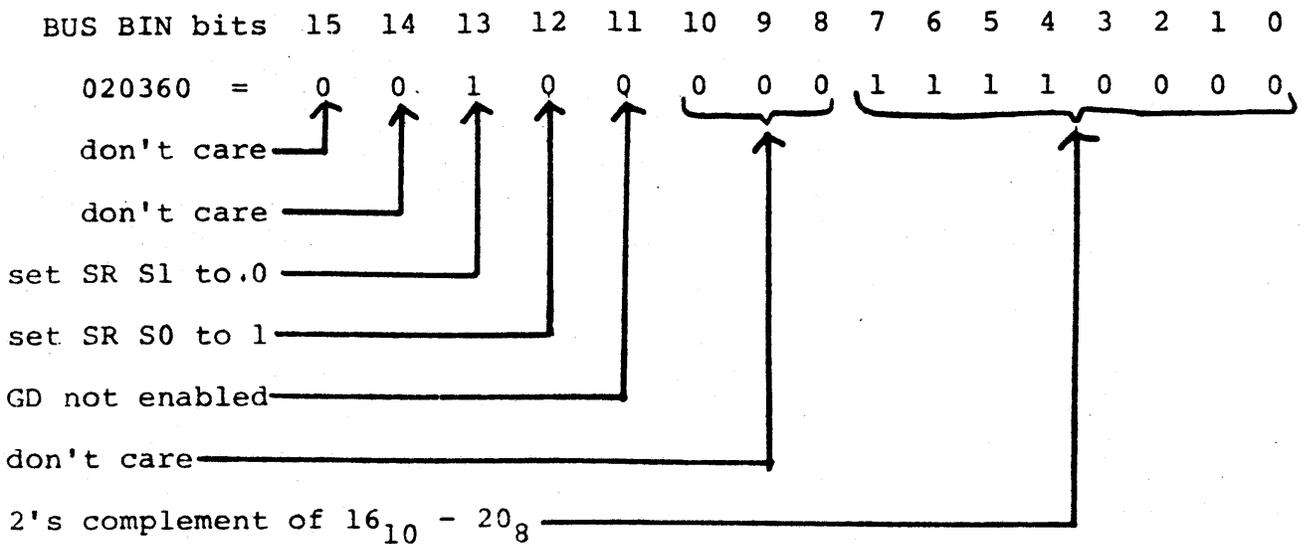
2.9.2.5 <u>Clearing RES</u> -- RES is cleared at P3 when a
BUT (CLEAR FLAGS) is issued. (BUT codes are described in
Section 3.xxx) Note that RES can be cleared also by loading
it.

Note that when the RES register is cleared its outputs default
to the following:

                SEND MUX S2 SEL (1) H --∅
                SR S1 (0) H -- 1
                SR S0 (0) H -- 1
                GD ENABLE (1) -- ∅
                GD ENABLE (0) -- 1

This means that the SR will be in parallel load mode, the
GD is not enabled, and the CMUX<00> input is either SR<15>,
D<07>, AMUX<03>.

2.10   Summary

This chapter has described the functional components of the 11/60 datapath. The fields shaded in Figure 2-36 control the components of the datapath. Table 2-12 summarizes the functions of these fields.

Table 2-12
Datapath Control Field Summary

Field Name	μword bits	Function
ALU	47:44	ALU function control
BEN	43:42	BUS BIN Enable
BSEL	41:40	BUS BIN address selection
AEN	39:38	BUS AIN Enable
ASEL	37:36	BUS AIN address selection
XMUX	36	XMUX port selection
CMUX	37:36	CMUX port selection
RIF	35:33	Immediate addressing of ASP & BSP; used in conjunction with ASEL, BSEL
COUT	32:30	COUT MUX selection (for D(C))
WHEN	29	P2 or P3 clocking
CLKD	28	Clock D register
CLKSR	27	Clock SR register
CLKBA	26	Clock BA at P1
CCC	25	Clock Condition Codes at P2 of NEXT μcycle
BMUX	23	BMUX port selection
AMUX	22:20	AMUX port selection
CSPADR	23:20	Complement of arbitrary address in CSP

Table 2-12 (cont.)

Field Name	μword Bits	Function
WRCSP	19	Write CSP at P3
HILO (MOD=0)	18	HI or LO sections of SPADS
CLKRES (MOD=1)	18	Clock RES register
WRSEL	17	A or B address on writeback
WRSP	16:15	Write BSP, ASP, or both
CLKCNTR (MOD=1)	16	Load CNTR from BUS BIN Bit Steering.
MOD	1	=0, Scratchpad Writeback =1, RES, COUNTER

Figure 2-37 is a block diagram of the 11/60 datapath.

Figure 2-36 DATAPATH CONTROL FIELDS

BASE MACHINE

CSPAD

SHIFT TREE

EMITS LOADREG

RETURN PAGING

UCON

RES BITS BIT(MULT)

PROC UCON

RT/CACHE UCON

I/O UCON

ALU | BEN | BSEL//EN | ASEL | RIF | COUT | ... | CSPADR | BUS | ... | UBF | UPF

RETURN PAGE | NEXT PAGE

UCONL | UCONH

AMUX | CMUX | CSPADR

* Signal asserted low

Figure 2-37

M7974-0-1
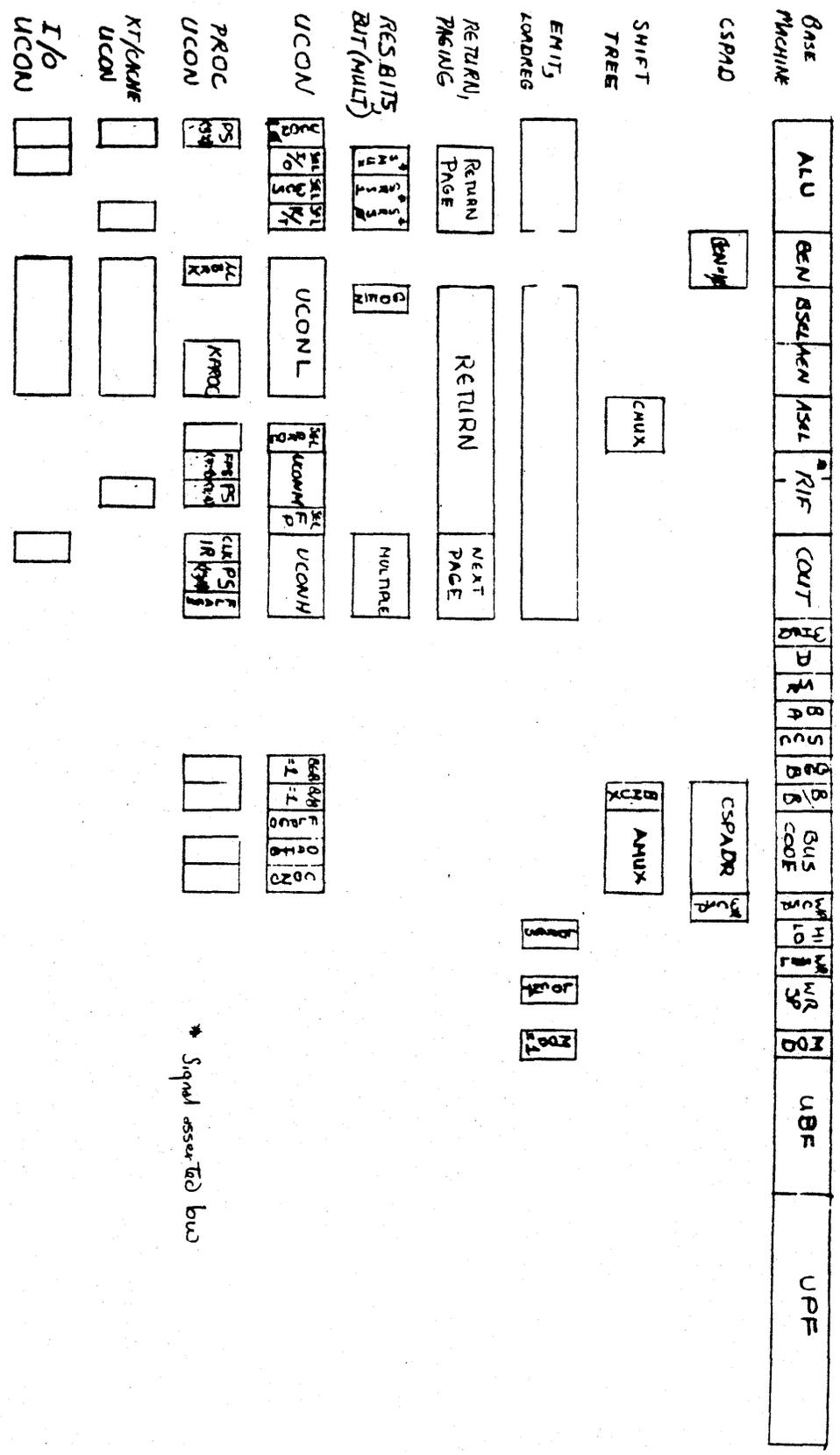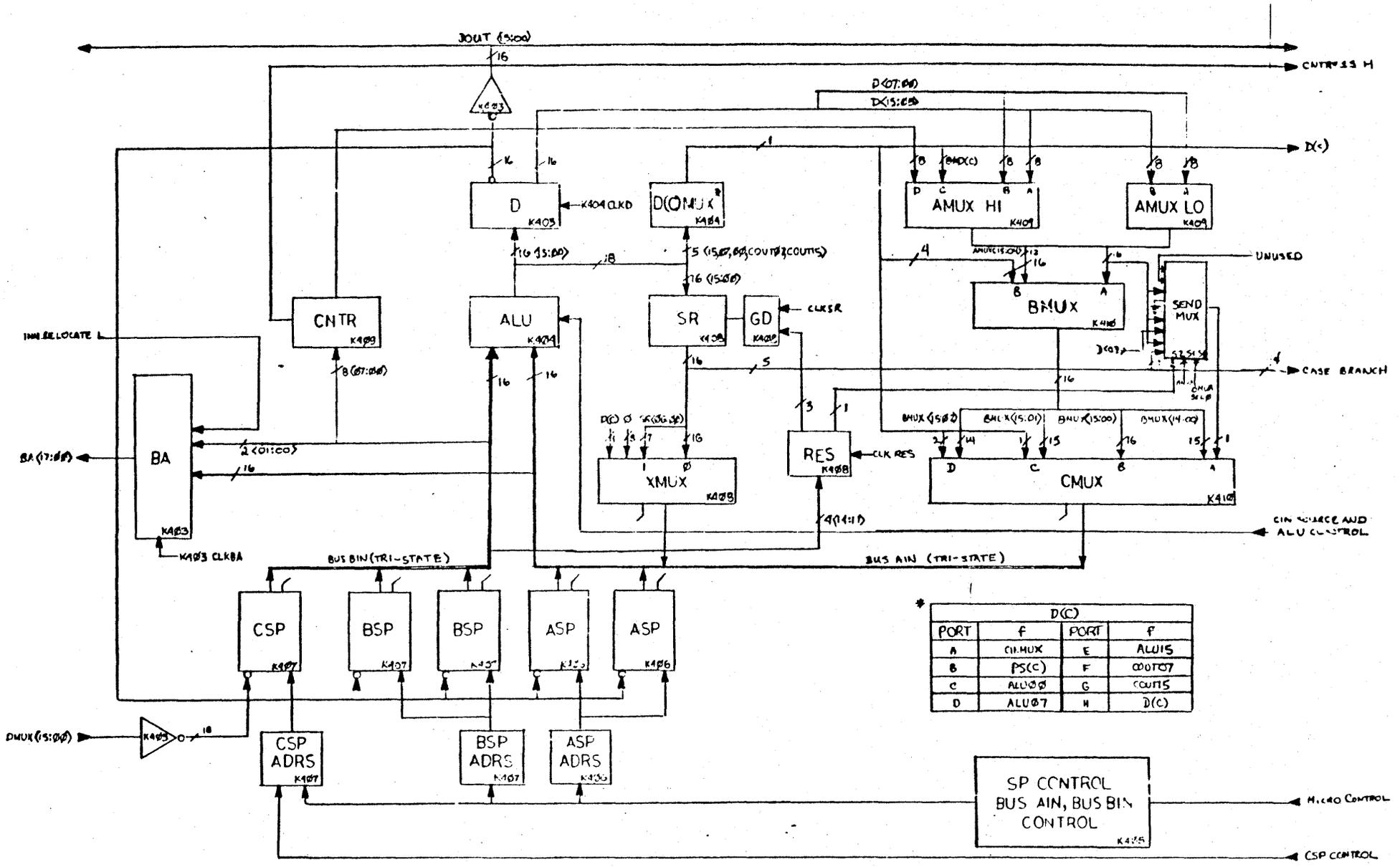
CHAPTER 3

MICROINSTRUCTION SEQUENCING

This chapter examines two aspects of microinstruction sequencing: address generation and the timing of microinstruction fetches.

3.1   CHAINED AND INSTRUCTION-COUNTER SEQUENCING

Two basic techniques for microinstruction sequencing exist, although they are used with many variations in different machines. We will call these methods chained sequencing and instruction-counter sequencing.

In chained sequencing, the current microinstruction contains the address of the next microinstruction. In this case, every microinstruction that is not a conditional branch is, in effect, an unconditional branch. This technique is derived from that originally proposed by Wilkes[1].

Instruction-counter sequencing is familiar to PDP-11 programmers. This method uses an incrementing microinstruction-counter register; microinstructions execute from sequential locations in the control store (with the exception of branches). In this second scheme, it is necessary to include an unconditional branch facility not required in the chained scheme.

---

[1]   Wilkes, M.V., The Best Way to Design an Automatic Calculating Machine, 1951

Both sequencing methods must make special provisions for
conditional branches. When a microinstruction contains the
address of its successor, it is common to include a field
in the microword to specify a test to be applied before the
next address is selected. Alternatively, a microword might
contain fields for two or more next addresses, selection
among them being made on the basis of conditions in the
machine. Incremental sequencing schemes may provide a field
for specifying a conventional two-way branch-on-conditon or
skip-on-condition opcode or may provide a facility to gate
the contents of a register into the microinstruction
register, thus replacing the sequentially generated address.

Selection of a sequencing method is based primarily upon the
organization of the microword and the micro-level architecture
of a machine. When there is a high degree of parallelism in
the datapaths of a system, and very few microinstructions
may be required to execute a single macro-level instruction,
the incidence of unconditional branches is high, and the
chained sequencing scheme is more efficient.

For these reasons, the 11/60 uses a chained sequencing method.
The MicroPointer Field (UPF) of the microword contains the
address of the next microinstruction to be executed. A
microprogram forms a chain, similar to a linked-list
data structure, as shown in Figure 3-1. The address specified
in the UPF field can be modified before it is used to select
the next microinstruction.

Before proceeding to a detailed discussion of branching, you
must look again at the issue of timing.

```
┌─────────┬─────┐      ┌─────────┬─────┐             ┌─────────┬─────┐
│         │ UPF │──►    │         │ UPF │──►  • • •  ─►│         │ UPF │──►
└─────────┴──●──┘      └─────────┴──●──┘             └─────────┴──●──┘
```

POINTER To
FIRST μ-instruction
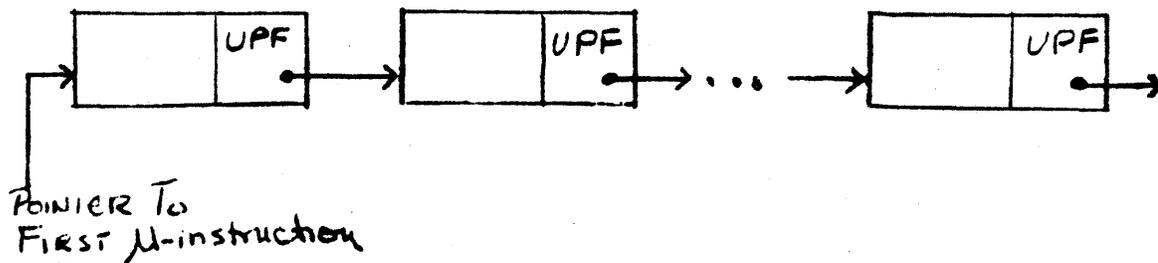
Figure 3-1   Chained Sequencing

3.2   TIMING

Section 1.5.2 stated that the 11/60 uses a clock with three
outputs, or pulses: P1, P2, and P3. An additional time point,
uP3, follows P3 by a few nanoseconds. This section examines
how those clock pulses are used, in combination with control
signals from the microword, to cause state changes in the 11/60.

In general, the clock pulses are used to tell a memory device
to load itself with the data currently at its input. In some
cases, the clock pulse signals the device directly, so that
the device is loaded every time the pulse occurs. In other cases,
the device is loaded at a clock pulse only if some other
caondition alqos exists; for example, the D register requires
CLK D from the microword as well as P2 or P3.

The memory devices in the 11/60 have a variety of names -- registers,
scratchpads, flip-flops, latches, etc. The type of loading signals
required by these devices divides them into two major groups:
those which are loaded on the edge of a pulse; and those for which
the input signal must be a level asserted over some period of time.

3-3

3.2.1   Control Timing

As previously stated, a new microinstruction takes control of the
11/60 every microcycle. The timing associated with the fetching
of microinstructions determines the control timing of the 11/60.

3.2.1.1   Fetch Timing

The terms serial and parallel, or non-overlapped and overlapped,
can be used to characterize when instruction fetches take place.

In the serial, or non-overlapped fetch, the next microinstruction
is not fetched until the current microinstruction is completed
(see Figure 3-3). This ensures that all information required to
select the correct microinstruction is available before the fetch
occurs.

The parallel, or overlapped system fetched the next microinstruction
while the current microinstruction is executing. This method has
obvious speed advantages, but can have problems handling
conditional branches. If the chioce of the next address depends
upon information generated during the execution of the current
microinstruction, the overlapped fetch will obviously fail.

Because the 11/60 takes advantage of the faster overlapped fetch,
careful attention to timing constraints when using conditional
branches will avoid unexpected loss of control.

a. Serial Fetch:

```
|     F1   |      E1      |
                          | F2    |    E2      |
```
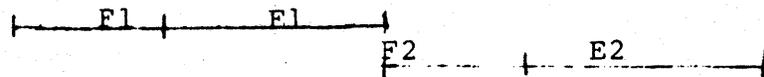
b. Overlapped Fetch:

```
|    F1  |      E1      |
              | F2   |    E2      |
```
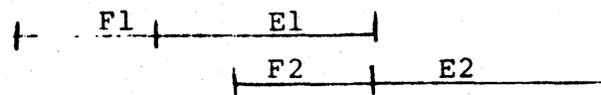
Figure 3-3 Fetch Timing

3-4

3.2.1.2 A Model for 11/60 Control Timing -- A very simple,
conceptual model for the control timing for the 11/60 is shown
in Figure 3-4. It consists of a microword register, branching
logic, and a control store.

The microword register is an edge-loaded memory device. Its
loading signal is generated unconditionally by uP3; that is,
it is loaded every time uP3 occurs. Its input data is the output
of the control store, which may be modified by machine state.
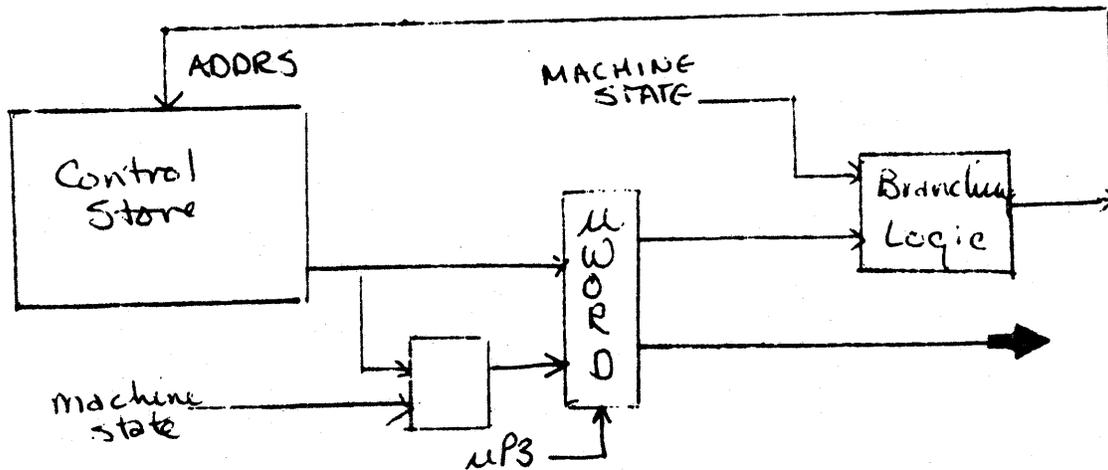
Figure 3-4; Control Timing Model

The output of the microword register basically controls the
actions of the 11/60 until the next uP3. Part of the microword
register is directed to the branching logic, along with some
machine state; the output of the branching logic selects the
next address in the control store.

Note

Figure 3-4 is a conceptual model only, it
does not represent the actaul control structure
on the 11/60.

3-5

Some of the control signals which come from the microword must
be held constant through two microcycles. Cases where this is
necessary are discussed elsewhere.

3.2.2 Intra-cycle Timing

The primary constraints on intra-cycle timing come from the
makeup of the basic computational loop of the 11/60 datapath.

The BSP and ASP are each composed of 32 level-loaded memory
devices, each loaded by different signals generated conditionally
from P3. Since the scratchpad must be enabled for loading
(writing), as well as for reading, the data on the corresponding
bus becomes undefined while the scratchpad write takes place.
This means that, for examples, when a location in the ASP is
being written, the data on BUS AIN is undefined for the time
period starting just after the leading edge of P3 until just
after the trailing edge of P3.

The ALU is a combinational logic element, whose output is a
binary function of its inputs. Even if the ALU function selected
is a unary operation such as "Select B", both inputs to the
ALU must be defined to produce the expected output.

The D register is an edge-loaded memory device whose loading
signal is generated conditionally at P2 or P3.

This information about the datapath shows that the following
operation can be performed in one microcycle:

$$P2, \quad D \leftarrow ASP[n] \quad PLUS \quad BSP[n]$$
$$P3, \quad ASP[n] \leftarrow D$$

Look at another operation, which at first glance seems feasible:

$$P3, \quad D \leftarrow BSP[n] \quad , \quad ASP[n] \leftarrow D$$

The intent is to move data from the B scratchpad into D while
moving the previous contents of D into the A scratchpad. Since
D is not loaded (and thus its output does not change) until the
trailing edge of P3, the constraint imposed by the level-loaded
scratchpad is satisfied. However, while the ASP is enabled for
loading, the BUS AIN input to the ALU is undefined. Hence the
ALU result is also undefined during that period, and the
correct result will not be loaded into D at P3.

The Shift Register (SR) is an edge-loaded memory device whose
loading is conditionally generated by P2 or P3. When functioning
as a shift register, the shift takes place, like the load, when
the trailing edge of the pulse occurs. Whenever both D and SR
are signalled in the same microcycle, their signals must be
generated from the same clock pulse. This prevents such operations
as :

$$P2, \quad SR \text{ LEFT } 1$$
$$P3, \quad D \leftarrow SR$$

However, it is easy to see that operations such as the following
are possible:

$$P2, \quad D \leftarrow SR \quad PLUS \ BSP[n]$$
$$P3, \quad ASP[n] \leftarrow D$$

or:

$$P2, \quad SR \leftarrow BSP[n]$$

Note that the ALU delay is slightly longer for arithmetic
operations than it is for logic operations.

3-7

3.2.3 Inter-Cycle Timing

Due to both the physical and logical structure of the 11/60, operations on sections of the processor other than the datapath generally require more than one microcycle (and hence more than one microinstruction) for completion. A number of factors affect considerations of timing across successive microcycles.

3.2.3.1 Memory Operations Timing

Data from memory is introduced into the datapath through the C scratchpad. This scratchpad can be loaded only at P3. The virtual address for a memory operation comes from the BA, which can be loaded at Pl.

The cache and memory management logic take a finite amount of time to process a request. Thus, even if the requested data is in the cache, there is not enough time between the trailing edge of Pl (when the BA is loaded) and the leading edge of P3 (when CSP data must be stable) for the data to be fetched. The loading signal for the CSP must be delayed until the next microinstruction. This situation is illustrated in Figure 3-5.

3-8
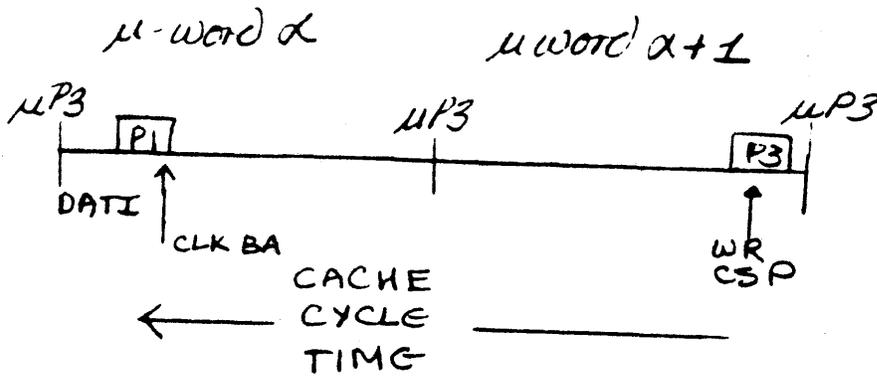
Figure 3-5: Requested Data in Cache

If the data is not in the cache, it must be fetched from main
memory. Since the main memory cycle time is much slower than that'
of the cahce, the data cannot be ready at the CSP input within
the normal time.

This asynchrony is handled by the generation of a "Pulse Supress"
signal. This signal is generated when the requested data is not
found in the cahce, and it prevents the generation of any clock
pulses until the Unibus cycle is completed. Figure 3-6 shows how
the pulse supression affects a data fetch from memory.
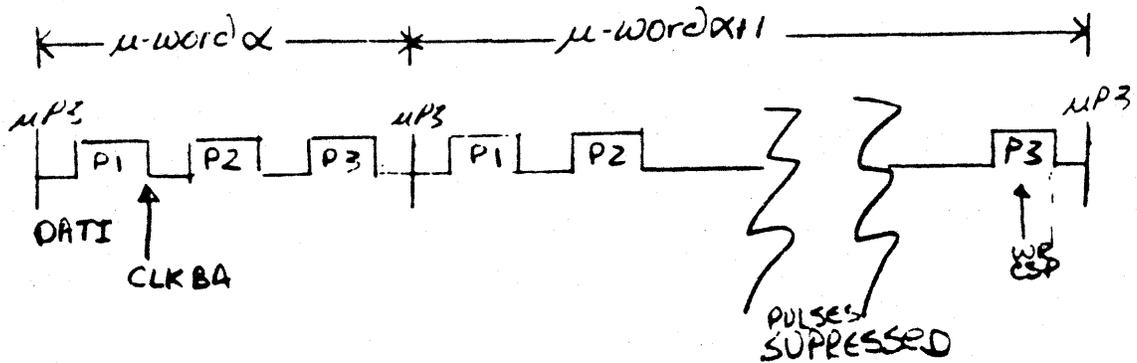
Figure 3-6   Data Not In Cache

3-9

When the memory reference is to an internal location, the

interrupted cycle should do nothing except clock the CSP, because

it will be executed twice. The base machine's JAM flow is used

to detect and service references to UNIBUS addresses located within the

processor: clocks are not supressed and datapath state is destroyed.
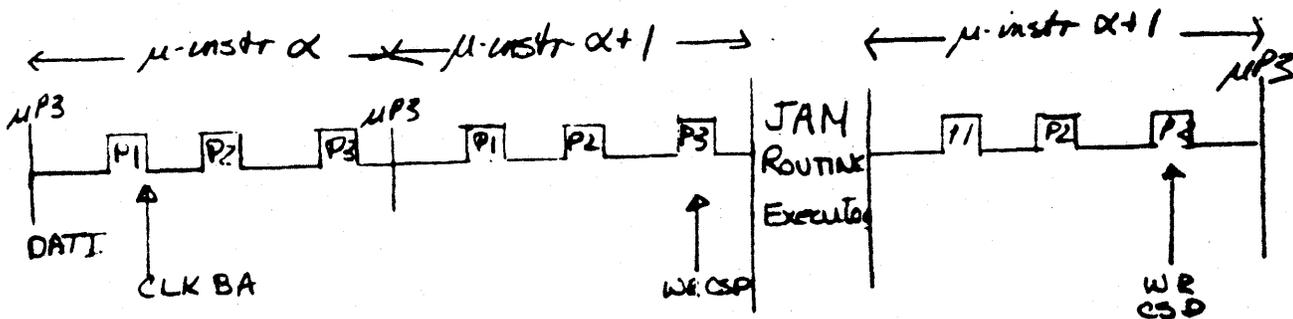
Figure 3-7   DATI, Internal Address

Every "DATA OUT" to a valid Unibus address involves a Unibus

cycle as well as a cache cycle. The clock pulses are supressed

after P2 of the microcycle following the DATO specification.

The cache update, with the address specified by the BA register, and

the data specified by the contents of the D register, begins at P3

of the first microcycle. The Unibus cycle does not begin until

after P2 of the second microcycle. Hence, the data to be written

must be clocked into D by P2 of the first microcycle and kept constant

until P3 os the cycle following the DATO. The procedure for doing

DATOs to valid Unibus addresses is shown in Figure 3-8.
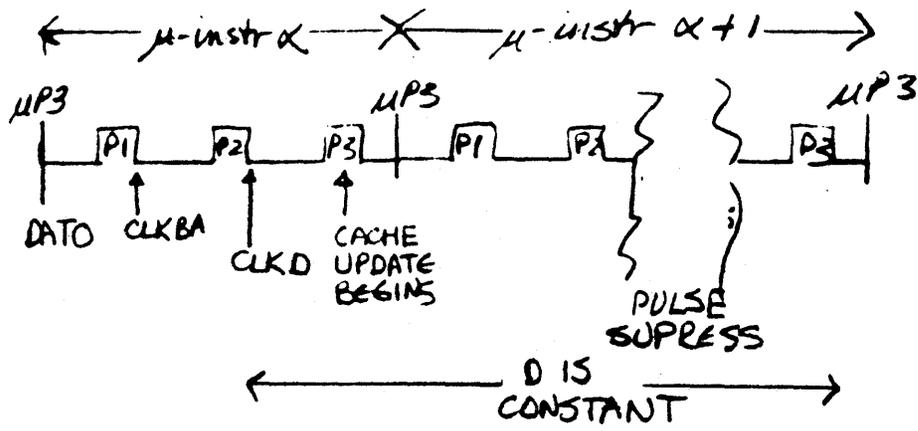
Figure 3-8   DATO Timing

A DATA OUT to an internal location looks very much like the corresponding DATA IN, as shown in Figure 3-9. The microinstruction following the DATO should be a null word. During the JAM routine, datapath clockings are repeated.
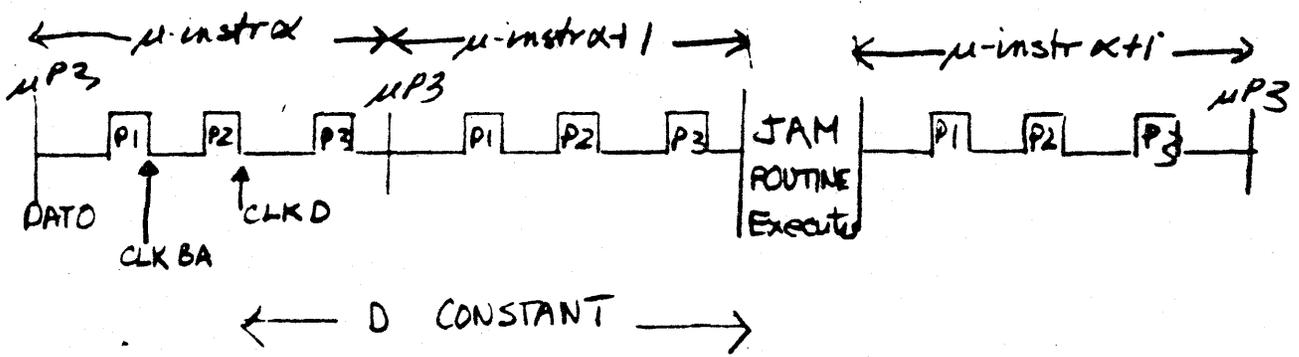
Figure 3-9   DATAOUT, Internal Address Timing

Since every memory reference requires two microcycles to complete, memory references MUST NOT be specified in two successive micro-instructions.

3.3   MICROCODE BRANCHING

This discussion of branching looks at two fields in the
microword. The MicroPointer field (UPF), μ<08:00>, contains
the address of the next microinstruction, as explained in Section
3.1. You can modify this sequence by using the MicroBranch
field (UBF), μ<13:09>. The UBF field serves three purposes:
1) it provides for conditional branches based on the state of
the machine; 2) it provides for subroutine calls and returns;
and 3) provides extra code-points for control signals.
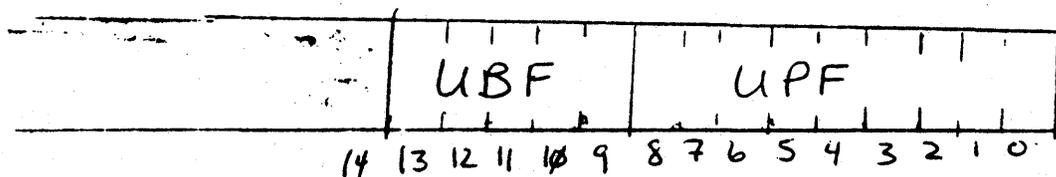
Figure 3-10   UBF, UPF Fields of the μword

  The mechanism for modifying the next microaddress is quite
simple. In the Processor Control Section, the Next MicroAddress,
which is used to address the control store, is generated by
ORing' the contents of the UPF field with the output of the BUT
MUX. The UBF field provides the selection signals for the
BUT MUX, as shown in Figure 3-11. The data inputs to the BUT
MUX are various elements of machine state, such as the
contents of SR<03:00>.

1) Actually a NOR gate is used; UPF bits must be
   inverted (∅=true) before storing them in the
   μCS. Array
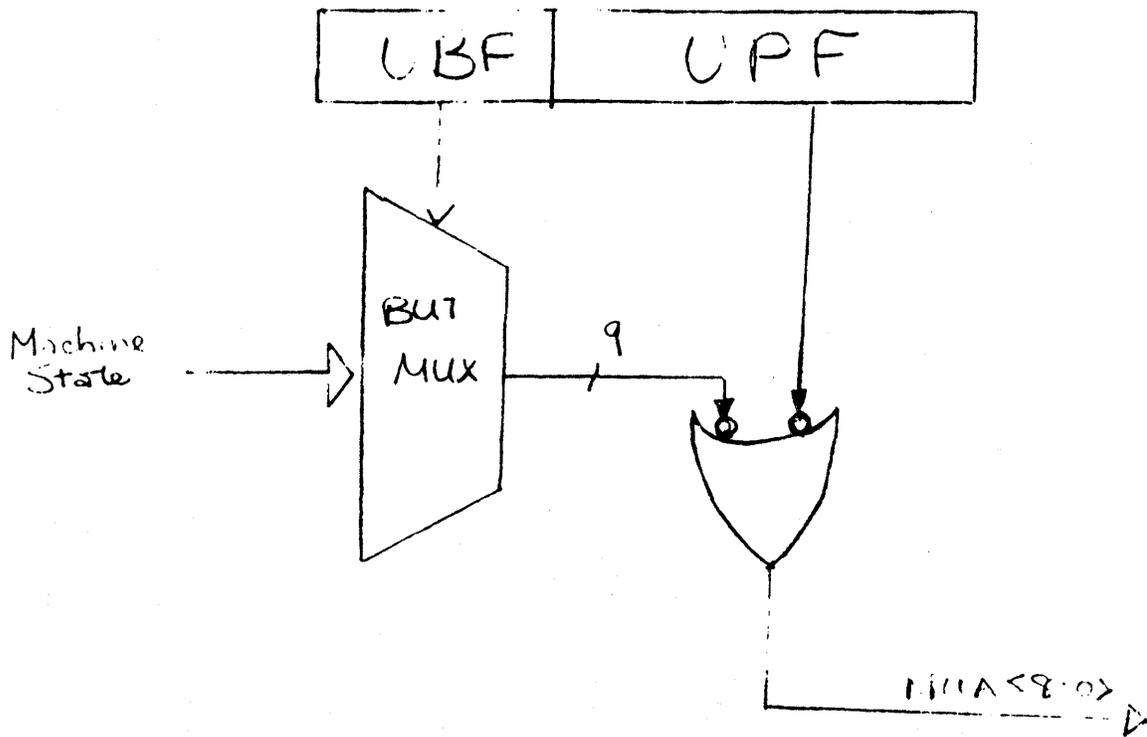
Figure 3-11   NUA Generation

When an unconditional branch (normal sequencing) is specified,
the output of the BUT MUX is all 0s. The ORing operation
does not modify the UPF, and the contents of the UPF field
becomes the NUA.

In conditional branching, the binary value ∨ the UPF field
is important because a UPF bit with a   : of 1 is not
affected by the ORing operation.

For example, consider a two-bit-wide branch, in which two
signals (s_1, s_2) are to be OR'd with the two low-order bits of
the UPF field. Potentially, this is a four-way branch. But if
either or both of the low-order UPF bits is a 1, the number
of potential target addresses is decreased. Figure 3-12

illustrates this effect. Note that you can use this to mask
out a signal in which you are not interested, as well as to
decrease the range of a branch.

                                          Potential Target Addresses

A.   UPF   x x x 0 0   x x x 0 0
                                     x x x 0 1
                    $s_1 s_2$ x x x 1 0
             _____   x x x 1 1
      NUA   x x x $s_1 s_2$

B.   UPF   x x x 1 0
                                     x x x 1 0
                    $s_1 s_2$
                                     x x x 1 1
             _____
      NUA   x x x 1 s_2

C.   UPF   x x x 1 1
                                     x x x 1 1   The state of the
                    $s_1 s_2$ signals has no
             _____   effect upon the NUA.
      NUA   x x x 1 1

Figure 3-12 Microaddress Modification

3.3.1   BUTs

The UBF codes, which control the branching logic, are given the
generic name BUT, for Branch Micro-Test. Since not all of the
code-points available in the UBF field are needed for conditional
branches, the BUTs are divided into two groups. The "regular"
BUTs only cause the ORing of the BUT MUX output with the UPF. The
Active BUTs as well change some micro-level state as well
as causing the ORing operation. Some active Buts perform a Null
But by ORing in only zeroes.

3-14

An example of an Active BUT is BUT(COUNT)(UBF/25). This
BUT is used both to increment and to test the contents of
the CNTR (see Section 2.6). Every time BUT(COUNT) is
specified, the CNTR is tested for overflow; after the test,
the CNTR is incremented. If the CNTR contained all 1s when
the test was performed, a 1 is OR'd with the low-order bit
of the microinstruction's UPF field. This provides a branch for
exiting when a loop is completed. Figure 3-13 illustrates
how the use of BUT(COUNT) affects the flow through a loop.
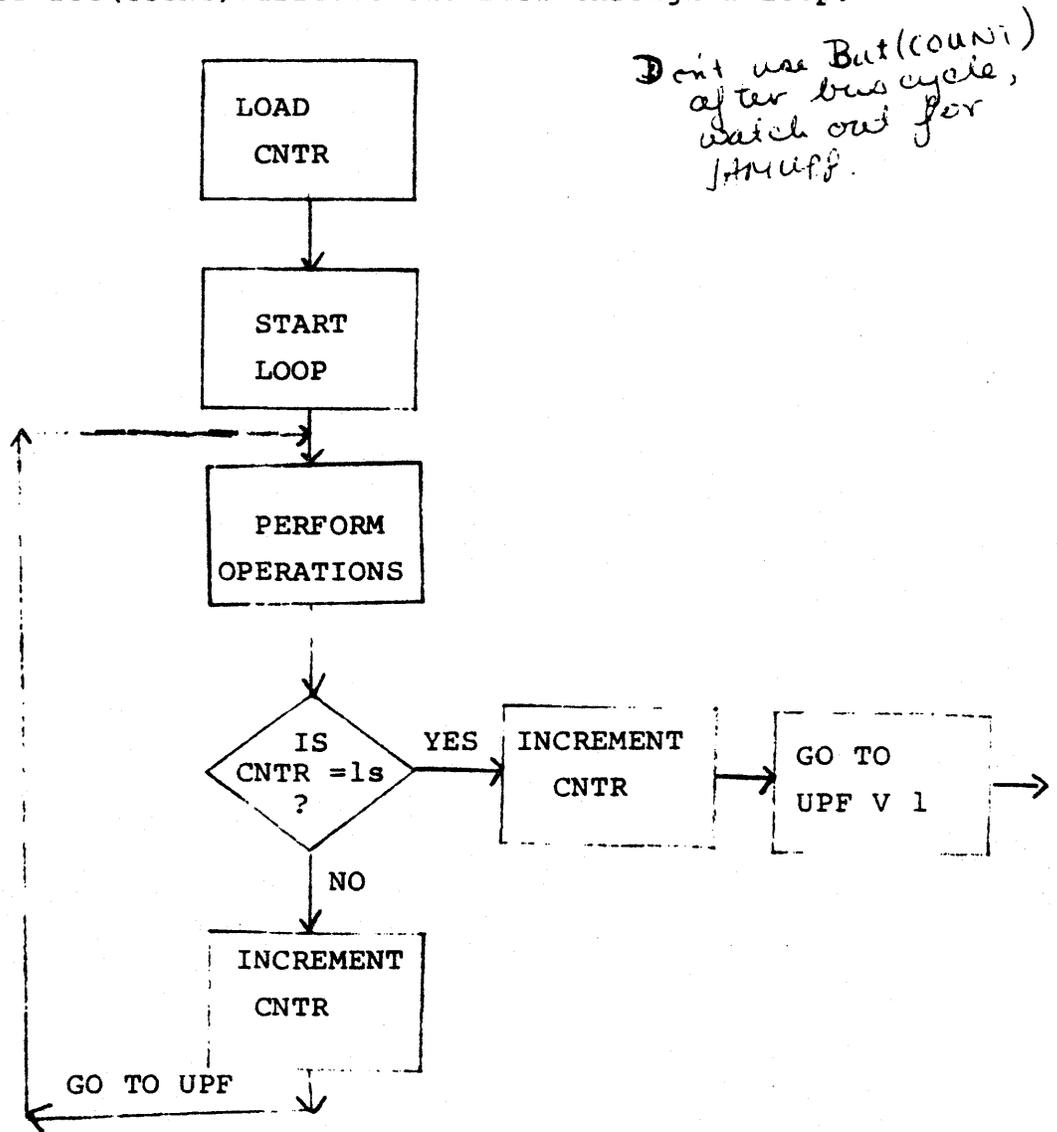
Figure 3-13   BUT(COUNT)

After the test, control will go either to the microinstruction at the address specified by the UPF field or to the micro-instruction stored at UPF OR 1.   .s that the first microinstruction within the **Loop** (pointed to by the UPF) is stored at ⟨an even⟩ location one less than the address of the first microinstruction handling the exit from the loop.

This example emphasizes the impact which chained sequencing has upon the programmer. In this case, the microinstructions within the loop <u>cannot</u> be stored in sequential locations.

3.3.2   <u>Timing Constraints on Branching</u>

Conditions to be tested in a branch must be set up in a microinstruction prior to the one which specifies the BUT code and the UPF base. ' NUA formation begins at μP3
                              ; thus, the machine state at the inputs to the BUT MUX is that which was clocked by the
   end   of the previous microcycle. On other words, conditions set up in microcyle 1 can be tested in microcycle 2 to affect the address of the microinstruction which controls microcycle 3, as   shown in Figures 3.14 and 3.15.

         μcycle 1   μcycle 2   μcycle 3
                            Hold Conditions

         Set-up   Test by   μinstruction Selection
         conditions   issuing BUT   depends on result
                              code   of ORing

         Figure 3-14   Setting Up Branch Conditions

μ-cycle 0

α_0 in control

μ-cycle 1

α_1 in control

μ-cycle 2

α_2 in control

μP3   μP3   μP3   μP3

Fetch of α_1

Fetch of α_2

Fetch of α_3

Clock μword

Clock μword

Clock μword

Form Address of α_1

Form Address of α_2

FORM Address of α_3

NEXT ADDRESS FORMATION
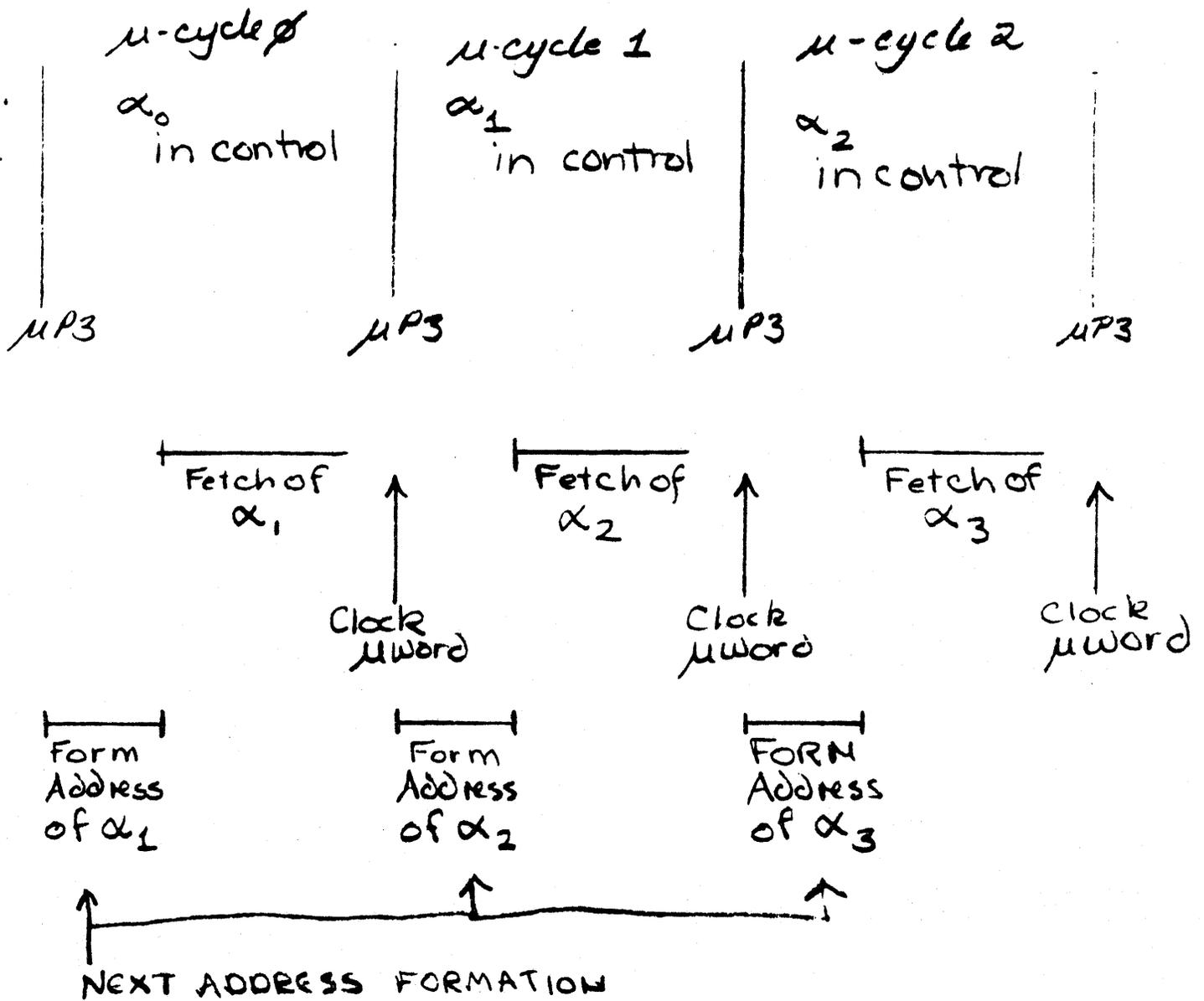
Figure 3-15   NUA = UPF V BUT MUX

The NUA is not clocked into a register before it is used to access the control store, so alterations in BUT conditions will interfere with ROM access. You should hold the tested condition stable throughout the microcycle in which the BUT code is issued. *However, you can change D at P3.* This is illustrated in the symbolic code below. This example uses BUT(DZERO) (BUT15), a 2-bit branch. The top bit, OR'd into UPF<1> is 1 when D<14:00> is all 0s; the bit OR'd into UPF<0> is D<15>.

```
    600:                        ! Arbitrary starting address
        P2:   SR ← R(DF)        ! Put data in D and SR at P2
        P2:   D ← R(DF)
            J/602               ! Go to next microinstruction
    602:
        P3:   D ←NOT SR         ! Complement data

            BUT(DZERO)          ! Test previous D data; if non-zero,
            J/601               ! go to 601; if zero, go to 603

    601:
        ! Continue

    603:
        ! Error return
```

The UPF specification (J/601) in the second microinstruction masks out the D<15> bit, so that only a test for D<14:00> = 0 is performed. The data stored into D in the first microinstruction is tested during the next, and the second clocking of D has no effect upon the branch.

3.3.3   The BUT List

Table 3-1 lists all the UBF codes (BUTs), their composition, and mnemonics. Many of the BUTs are described in detail in later sections of this manual;   the others . perform as indicated in the Notes column of the table. The shaded rows

indicate BUTs which are least likely to be of any use to the
WCS microprogrammer. These BUTs were designed specifically to
aid in the implementation of the PDP-11.

Notice that the NULL BRANCH, that is, the UBF code that causes
only 0s to be OR'd with the UPF and changes no state in the
machine is BUT30. The UBF is the only field in the microword
in which a value of 0 is not an acceptable default; *use the
value 30 as your default branch code*

All BUT conditions are active high; and all branch widths are
justified to the low order microaddress.

3.4   THE CASE BRANCH

The CASE branch, BUT(CASE), causes the four low-order bits of
the SR to be OR's into the UPF field.

Using BUT(CASE), control can be directed to any one of 16
locations within the current page. Thus, if the UPF field
contains the address 340_8 (011100000_2), the next micro-
address could be any one of the following:

340	344	350	354
341	345	351	355
342	346	352	356
343	347	353	357

You may not always need the full sixteen-way branch capability
of BUT(CASE). If the UPF address in the example above were
343_8 instead of 340_8, the branch could only go to *four*
locations: 343, 347, 353, and 357. Only the condition of SR<3:2>

Table 3-1

BUT LIST

UBF CODE	COMPOSITION (Bits OR'd into UPF)	WIDTH	NAME AND SYNONYMS	ACTION AND USAGE
ØØ	SR<03:00>	4	BUT(CASE)	See Section 3.4
01	IR<15:12>	4	BUT(DOP)	Decodes opcodes of double operand instrs.
02	INSTR 5 BR<4;0>	5	BUT(INSTR5)	Decodes opcodes
03	IR11◻ FLTPT BR<3:0>	5	BUT(FLPDECODE)	Floating point decode
04	IR<09:06>	4	BUT(SOP)	Decodes opcode of single operand instructions
05	Ø◻ MOV ∧ $\overline{FLPT}$ V DR7 ∧ FLPT◻ IR<05:03> 5			Indicates whether IR contains a MOV instr. or a floating point opcde with destination register 7 (PC); and what the Destination mode is.
06	INSTR 1 BR<7:0>	8	BUT(INSTR 1)	Initial PDP-11 instr. decode.
07	Ø◻ BGINTERNAL V FLPT SRVC◻ D(C) ◻FLTPT ACK◻ FRETØ	5		
10	COUTØ7◻ DOUTØ7◻ FBS05	3	BUT(FNORM)	Used in normalization in floating point
11	DMØ◻ SMØ◻ BYTE	3	BUT(DMØ◻SMØ◻BYTE)	Indicates whether current instr. has destination mode Ø, source mode Ø, and if it is a BYTE operation.

TABLE 3-1   BUT LIST (CONT.)

UBF CODE	COMPOSITION (Bits OR'd into UPF)	WIDTH	NAME AND SYNONYMS	ACTION AND USAGE
12	GUARD<3:2>	2	BUT(GUARD)	Top two bits of GD are OR's into UPF; use for checking results of shifts, etc
13	SR<01:00>□CNTR<7:0>=1's	3	BUT(MULSTEP)	Used in Multiply loop; tests CNTR, increments it, and indicates what is in SR 01:00
14	BGINTERNAL□ MF INSTR□MULTI BR	3	BUT(MFINC□MULTIPLE) BUT(MULTIPLE)	If you mask out the top two bits, can use BUT(MULTIPLE) See Sec. x.x
15	(D<14:00> = Ø's)□ D<15>	2	BUT(DZERO)	Indicates if D<14:00>is all Øs, and what D<15>is
16	IR11□ PS15	2	BUT(JMP, JSR) BUT(IR11□ PS15)	IR<11>distinguishes JMP and JSR instrs., PS<15> indicates current mode.
17	(CNTR<7:0> =1's)□ D(C)	2	BUT(ASHBR)	Tests and increments CNTR, shows what D(C) is.
20	NO INSTR OVERLAP □ SERVICE	2	BUT(FOV □ SERVICE) BUT(SERVICE)	Indicates no PDP-11 instr. fetch overlap; SERVICE checks for service conds.; must be performed before every PDP-11 instr.
21	PSSYN INTERNAL□ DR(6 V 7)	2		
22	Ø	ACTIVE	BUT(ROR1)	Low bit of IR source field is OR'd with 1; use to address multiple regs.

TABLE 3-1   BUT LIST (CONT.)

UBF CODE	COMPOSITION (Bits OR'd into UPF)	WIDTH	NAME AND SYNONYMS	ACTION AND USAGE
23	D(C) ■ BA<00>	2	BUT(D(C)◻ODD ADDRESS)	Indicates carry and odd address
24	OTHER JAMUPP ◘ FLAG05 ∧ $\overline{\text{EXFLAG01}}$	2	BUT(OTHERJAMUPP ◻ HOTWARM)	Used by base machine
25	CNTR<7:0> ALL 1s	1	BUT(COUNT)	Use to test and increment CNTR reg.
26	INTR REQ ■ NO BR INSTR	2	BUT(INTR REQ■SUCBRANCH)	
27	FOVLAP SAVE◘ FPS07	2	BUT (FOVPSAV ◻ FPS07), BUT(FD)	
30	∅	ACTIVE	BUT(NULL)	This is the NULL BRANCH; UBF and machine state unchanged
31	∅	ACTIVE	BUT(TRACK)	BUT(TRACK) enables CUA tracking, which is disabled upon JAMUPP
32	∅	ACTIVE	BUT(CLEAR FLAGS)	Clears RES register and Short Term Flags (MFPI, MTPI, T-Bit mask), Selects EMIT, Clears UCON
33	∅	ACTIVE	BUT(DIAGNOSE)	Reserved for DCS
34	∅	ACTIVE	BUT(SUBR B), BUT(GO TO)	Return<11:00> ← RETURN, Page<2:0> ← PAGE
35	∅	ACTIVE	BUT(SUBR A)	Loads Return and Page regs. Return<11:00> ← D<14:03> Page<2:0> ← PAGE
36	∅	ACTIVE		Loads NUA, Return, and Page Regs
37	∅	ACTIVE	BUT(RETURN)	Page <2:0> ← Return <14:0?> NUA <11:0> ← RETURN <08:00>, RETURN <14:0> ← D<14:03>

would affect the outcome of the branch.

A simple example that uses the CASE branch is testing whether
an ALU result is even or odd.(Obviously, there are other ways
of doing this.) To ensure that you test only the desired
condition, UPF<3:1> are 1s. The first microinstruction specifies the
ALU operation and clocks the result into the SR; the second
specifies BUT(CASE) and the base address for the branch.

```
    INSTR 1:
            P2:   SR ← A PLUS B,
                  BUT(NULL), J/INSTR 2
    INSTR 2:
                  BUT(CASE), J/NEXT      ! Go to NEXT if result
                                         ! is even, NEXT+1 if odd
```

Using the microassembler, you would specify a constraint field
of 1110 for NEXT.

3.5   SUBROUTINES

The well-proven programming technique of subroutine
structure is available to the IMP microprogrammer. Use of
subroutines generally results in smaller microprograms,
more systematic microprogramming, and more easily shared
microroutines.

3.5.1   BUTs for Subroutines

There are three BUTs to control entry to and exit from
microcode subroutines. They are BUT(SUBR B), BUT(SUBR A), and
BUT(RETURN). All are Active BUTs.

A jump to the beginning of a subroutine is distinguished from
an unrestricted jump by the storing of a return address
prior to the jump. On the IMP, the return address is stored
in the Return register in the Processor Control Section. The
jump is then made in the normal way to the location specified
in the UPF field. At the end of the subroutine, BUT(RETURN)
causes a jump to the previously stored return address by
loading the NUA with the contents of the Return register.

The return address can be loaded from the D register or
directly from the microword. You will generally load the
return address from the microword for first-level subroutines,
and load it from D when nesting or dispatching on previously extracted
bit patterns.

BUT(SUBR B) stores the contents of the RETURN PAGE and RETURN ADDRESS
fields of the microword, μ<46:44> and μ<41:3 >, in the
Return register. Note In Figure 3-16 that these fields ___ data path
overlap the ALU function field and some of the √bus control
fields. Hence, attempt no datapath manipulations in a
microinstruction that specifies BUT(SUBR B).

	RETURN PAGE		RETURN ADDRESS		
ALU	BEN	BSEL	AEN	ASEL	RIF
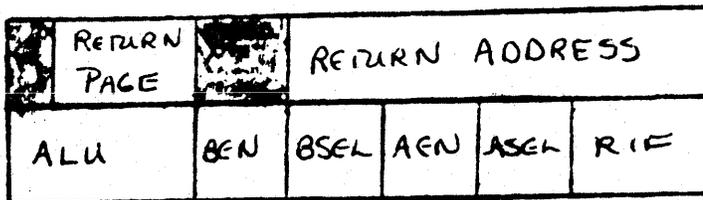
Figure 3-16   RETURN PAGE, RETURN Fields of μword

The following example shows how BUT(SUBR B) and BUT(RETURN)
can be used when no nesting is involved.

| MAIN FLOW | SUBROUTINE |

```
                     .                        SUB1:      .
                     .                                   .
                     .                        ENDSUB:    .
       JMP:                                       BUT(RETURN)
          RETURN ← MAIN1Ø,
          PAGE ← 1,
          BUT(SUBR B),
             J/SUB1
   MAIN1Ø:
```
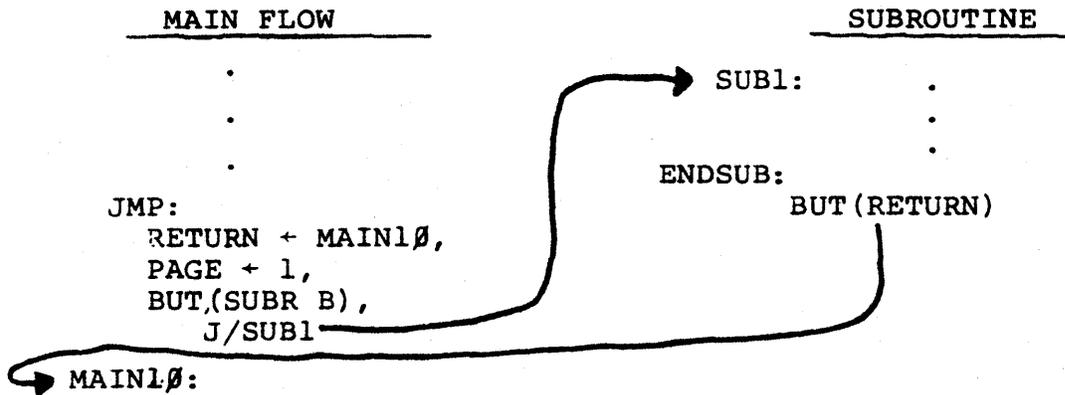
Both BUT(SUBR A) and BUT(RETURN) load the Return register
from D<14:03>. This allows you to use a previously calculated
or stored return address.

Although you can use BUT(SUBR A) and BUT(RETURN) for
general subroutine calling like BUT(SUBR B), they are
especially useful for nested subroutines. When going through
successive levels of subroutines, you must save the return
addresses in the scratchpads.

The following sample microcode illustrates the use of BUT(RETURN)
to take a return address out of the CSP.) Note that the caller
stores the return address.   ↳VIA EMIT

```
       MAIN:
          P3, CSPB[MD]←MAIN1           ! CSP GETS RETURN ADDRS
          PAGE ← 1,   BUT(SUBR B),
             J/SUB1

       MAIN1:                          ! RETURN ADDRS FOR SUBROUTINE 1
```

```
SUE1:                               !SUBROUTINE 1 CODE

    J/SUB1N

SUB1N:
    RETURN + SUB1M,                 !LOAD REUTRN ADDRS FOR SUB2
    BUT(SUBR B), J/SUB2

SUB1M:
    BUT(RETURN)                     ! RETURNS CONTROL TO MAIN1

SUB2:                               !ENTRY POINT
      J/SUB2N

SUB2N:                              ! THIS INSTR  SETS UP
    P2: D   CSP[MD],                !' D SO THAT RETURN WILL BE
    J/SUB2N+1                       !' CORRECTLY LOADED IN SUB2N+1

SUB2N+1:
    BUT(RETURN)                     ! THIS INSTR RETURNS
                                    !CONTROL TO MAIN 1 AND
                                    ! LOADS THE RETURN REG FROM
                                    ! D
```

The function of this code may be clarified by the flow diagram below.

3-26
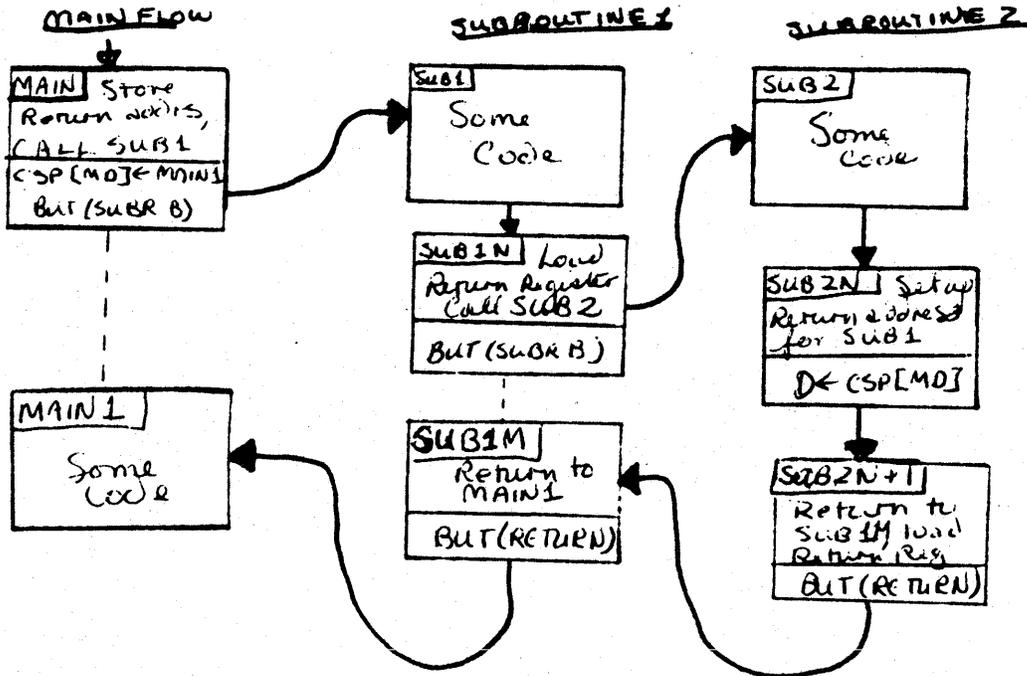
3.5.2 Using Subroutines

In structuring your microprograms, it is important to
understand how the chained sequencing scheme of the IMP
affects the use of subroutines. In higher-level languages
where the instruction flow is sequential, a subroutine
can be seen as a "black-box" process occuring between two
main-program instructions. That is, the subroutine process
is isolated, and is generally called from and returns
to the same program flow.

On the IMP, there is no automatic distinction between
in-line and called code because every microinstruction can
call any other microinstruction in the control store.
In addition, a microcode process can be isolated merely
by ensuring that entry to and exit from its flow occurs
only at specific points.

The most general approach to understanding and using the
microcode subroutine facility of the IMP is to consider it
   a method for sharing code sequences. Thus, if flows
A,B, and C contain common code, only one copy of the common
code is required. The main flows A, B, and C specify
BUT(SUBR B), thus loading the return address, in the
microinstruction that specifies the jump to the common
code sequence. The last instruction in the shared code
includes a BUT(RETURN), which   returns control to the
proper main flow.

Note that control does not have to return to the calling
flow. Within a subroutine, (you can specify) a conditional branch
            which can preclude the execution of the
BUT(RETURN).

3-27

3.6   PAGE CHANGING

Heretofore, we have looked only at the low nine bits of the microaddress. Twelve bits are needed to specify a unique control store location on the IMP.

The IMP control store is divided into 512-word pages. The 9-bit address selected by the UPF field represents the displacement within a page. Three additional address bits are used to specify the page.

Unnecessary page changing is to be avoided, since it can add overhead.

The top three bits of the microaddress are specified by the contents of the Page register in the Processor Control section of the IMP. The Page register is loaded whenever BUT(SUBR B), BUT(SUBR A), or BUT(RETURN) is specified. Thus, page changing can only occur when one of these BUTs is specified in the UBF field of a microinstruction.

The contents of the PAGE field of the microword, $\mu\langle 32:30 \rangle$, are loaded into the page register whenever BUT(SUBR B) or BUT(SUBR A) is specified. Do not confuse this field with the RETURNPAGE field, $\mu\langle 46:44 \rangle$; refer to Figure 3-17.

When BUT(RETURN) is used, the top three bits of the Return register are loaded into the Page register. (Note that the original source of these bits was the RETURNPAGE field when BUT(SUBR B), BUT(SUBR A), or BUT(RETURN) was specified.)

(was specified)

(and D$\langle 14:12 \rangle$ when)

3-28
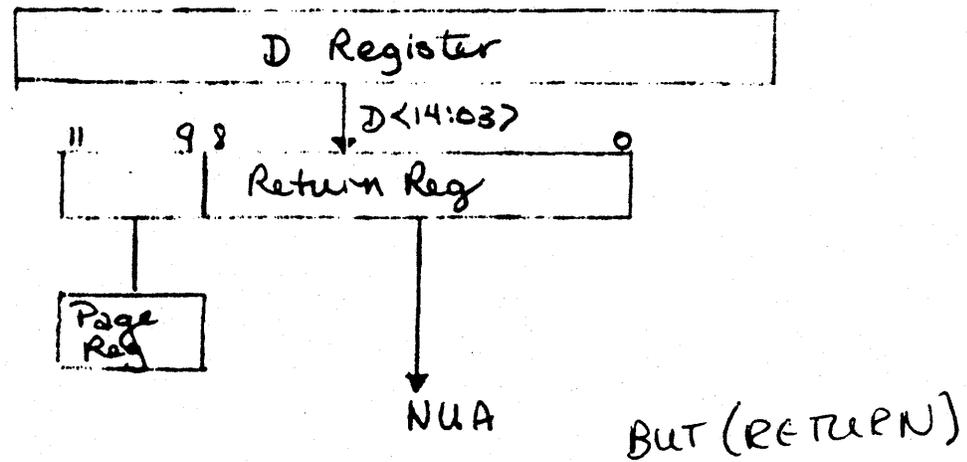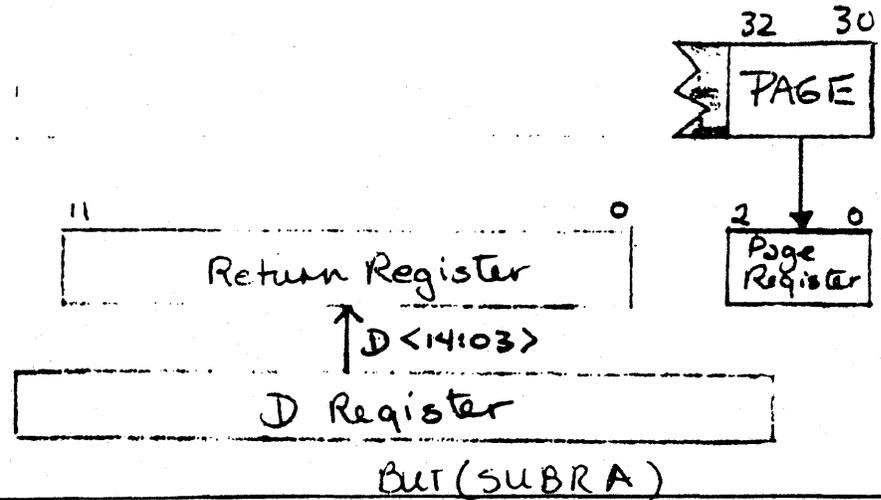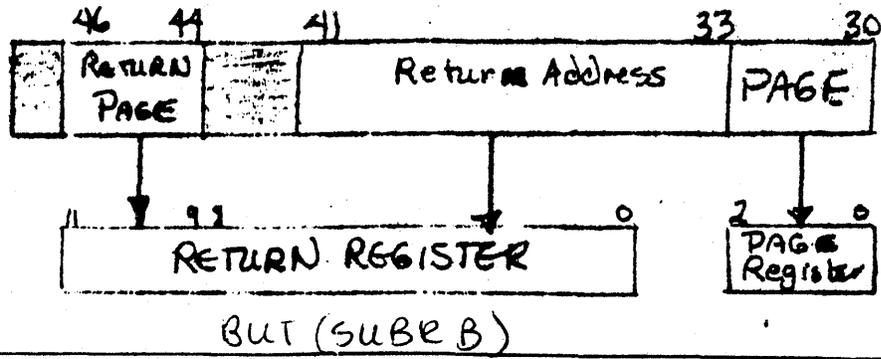
Figure 3-17   Three Ways of Loading the Page
                Register

These three BUTs were discussed in the context of subroutining,

but it is important to note that oage changing is not
restricted to subroutine dispatching. Although the three BUTs
load the Return register, as well as the Page register, that
loading is significant only if a BUT(RETURN) is issued.

Thus, if the microinstruction at location 6056 (page 6,
location 56) looks like this:

```
        6056:
                PAGE/7, BUT(SUBR B), J/220
```
the next microinstruction will be at 7220 (page 7, location
220). The dat loaded into the Return register by the
BUT(SUBR B) in 6056 doesn't matter unless 7220, or one of
its successors, specifies a BUT(RETURN).

The following is an example of changing pages while calling
a subroutine:

```
        6056:
        CALL:
              RETURNPAGE/6, RETURNADDRESS/057, PAGE/7,
              BUT(SUBR B), J/220

        7220:
              BUT(RETURN)

        6057:
```

Because the subroutine BUTs are also the page-changing BUTs,
there is a danger of jumping off the current page inadvertently.
You can avoid this problem by making sure that microinstructions
using BUT(SUBR B) and BUT(SUBR A) have the proper value in
the PAGE field. A normal subroutine must not change pages,
because it would destroy the contents of the RETURN register.
(Note that AND operations are possible while changing
pages, but since PAGE overlaps COU, DC, D
will be random if D is clocked during the
microinstruction.)

3-30

CHAPTER 4

THE CENTRAL PROCESSOR

While you will work almost exclusively with the ||/60 Inner
Machine (the Datapath and Processor Control sections),
potentially useful features exist in other sections of the
processor. In addition, it is important to be familiar with
the inter-relationships of the various sections of the
processor.

The fold-out block diagram of the processor, Figure 4-
will be a useful reference while reading this chapter.

4.1 INTRA-PROCESSOR COMMUNICATION

The Datapath can send data to, and receive data from, each of
the other four sections of the processor. The following sections
discuss the mechanism for these data transfers, and the means
of controlling them.

4.1.1 BUSDIN and DOUT

BUS DIN and   DOUT connect all the sections of the IMP
processor, and are the main data channels within the
machine. Both are 16 bits wide.

The only device that can put data on BUS DOUT is the D
register. This data is then available to all other sections of
the processor. No explicit enable signal is needed to put
the contents of D onto   DOUT. Thus, if the contents of D
are unstable, so is   DOUT.

BUS DIN supplies data to the Datapath: every section of the
processor <u>except</u> the Datapath can put data on BUS DIN.
BUS DIN is connected to the DMUX (which provides the CSP input).
Hence a WR CSP specification is needed to get BUS DIN data
into the datapath. All the other sections of the processor
have tri-state multiplexers connected to BUS DIN. Selection
and enabling of these multiplexers is controlled by the UCON
register.

CAUTION
Improper use of UCON can cause hardware damage.

4.1.2   UCON Control Register

The UCON register is a 16-bit set-up register, located in the
Processor Control section. (The general concept of set-up
registers is explained in Section 2.8.) UCON controls intra-
processor communication, that is, micro-level data transfers
between sections of the I/μC processor.

The contents of the UCON register determine which section of
the processor is to be accessed from the datapath, and in
what manner. It is simplest to look at the UCON register in
two parts, according to the function of the bits. The low-
order five bits control the selection of a section of the
processor and any necessary enabling. The remaining 11 bits
provide further control of the section selected.

The UCON register is loaded at μP3 whenever BGB, μ<24>,
BUSBOX, μ<23>, and CONO, μ<20>, are all equal to one. Figure
4-1 shows how the UCON register is loaded from the micro-
word. Since the interpretation of the control bits depends upon
the section of the processor selected, their functions are not
shown in this illustration.
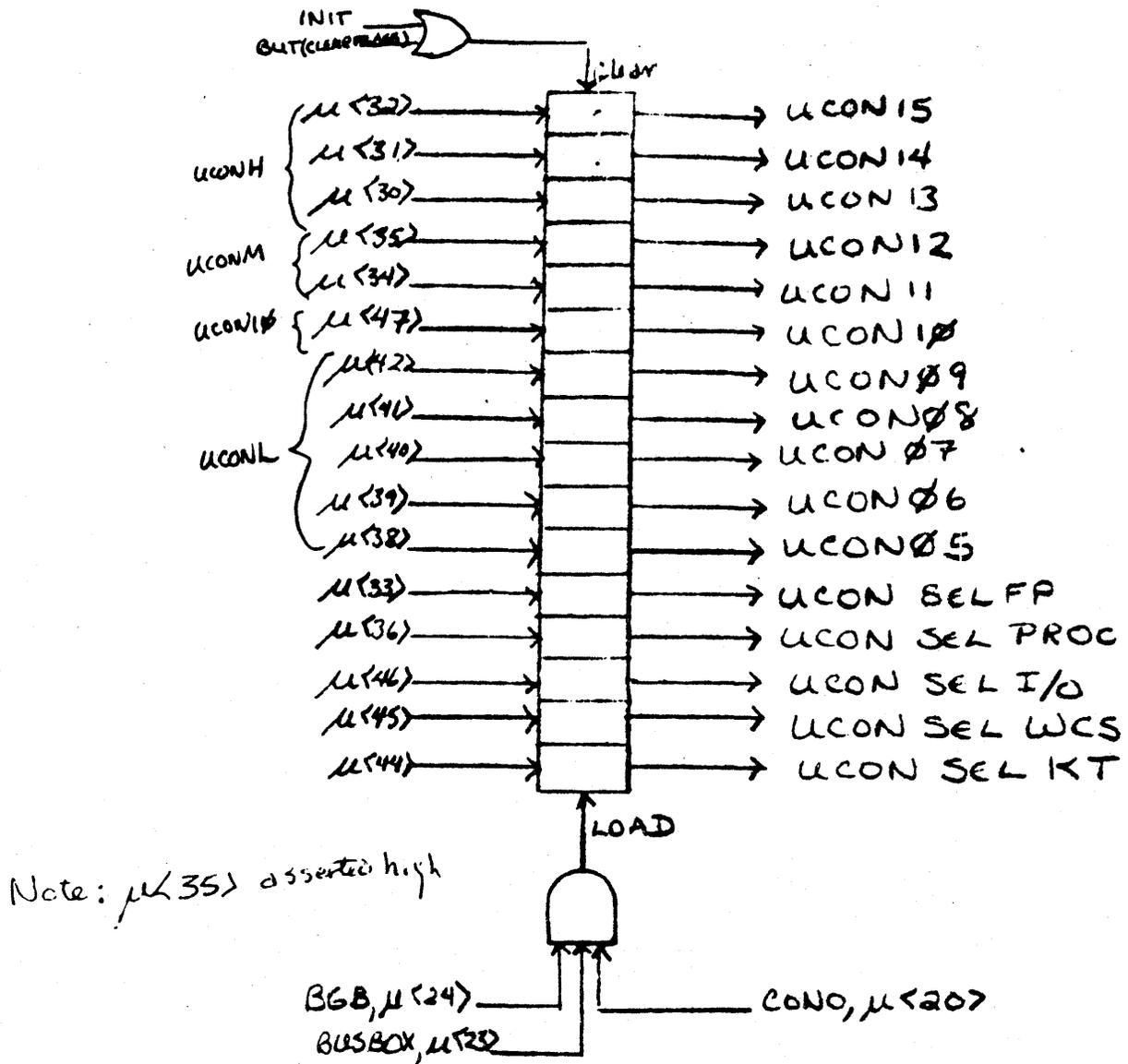
Figure 4-1   UCON Register

The order in which the bits are loaded into the register is
transparent to the microprogrammer.   It is the mapping of
microword fields to eventual effects (signals) that impacts
the microprogrammer.

The UCON register sets up a routing path for data whenever an intra-processor bus cycle is specified by the microprogrammer. Thus, when writing of datapath data is specified, UCON determines which section(s) of the processor is to take data off BUS DOUT, and to which register in that section the data is written. When data is to be introduced into the datapath, UCON provides the appropriate enable and disable signals to the tri-state multiplexers attached to BUS DIN.

Once this path is set up, it does not change until you reload UCON. You can set up UCON before it is needed, and use it repeatedly until the register is reloaded.

Specific details of the UCON interface to each section of the processor, such as the function of the control bits, are included in the remaining sections of this chapter.

4.1.3   UCON Control Fields

The BUS/UCON control fields, which span $\mu<24:20>$, serve to distinguish between and provide control signals for both Unibus cycles and Intra-processor data transfers. Their use for Bus (Unibus) control is described in Section 4.3.

The BGB field (think of it as Begin UCON or Begin BUS), $\mu<24>$ determines if activity over BUS DIN or DOUT is going to occur. If BGB is equal to 1, bus activity is allowed. This bit avoids inadvertent bus cycles when setting up the

Shift Tree or addressing the CSP due to the overlapping bit fields. A BGB value of 1 indicates that either a Unibus cycle or UCON activity is going to take place.

BUSBOX, μ<23>, determines how the remaining bits are to be interpreted. If BUSBOX is 1, then μ<22:20> are used in controlling intra-processor communication.

FLTPT, μ<22>, is used by the floating point hardware and should always be 0 when performing UCON activity from the WCS control store.

DATTB, μ<21>, is used in different ways by different sections of the processor.

CONO, μ<20>, is used in conjunction with BGB and BUSBOX to load the UCON register.

4.2   THE INNER MACHINE

The Inner Machine is composed of the datapath and the Processor Control sections. This section describes the features of the Processor Control section and its interactions with the datapath.

The Processor Control section, as well as providing control signals to the datapath, contains a number of important data registers. Understanding how these registers can be accessed from the datapath will give you added flexibility, both at the macro- and micro-code levels.
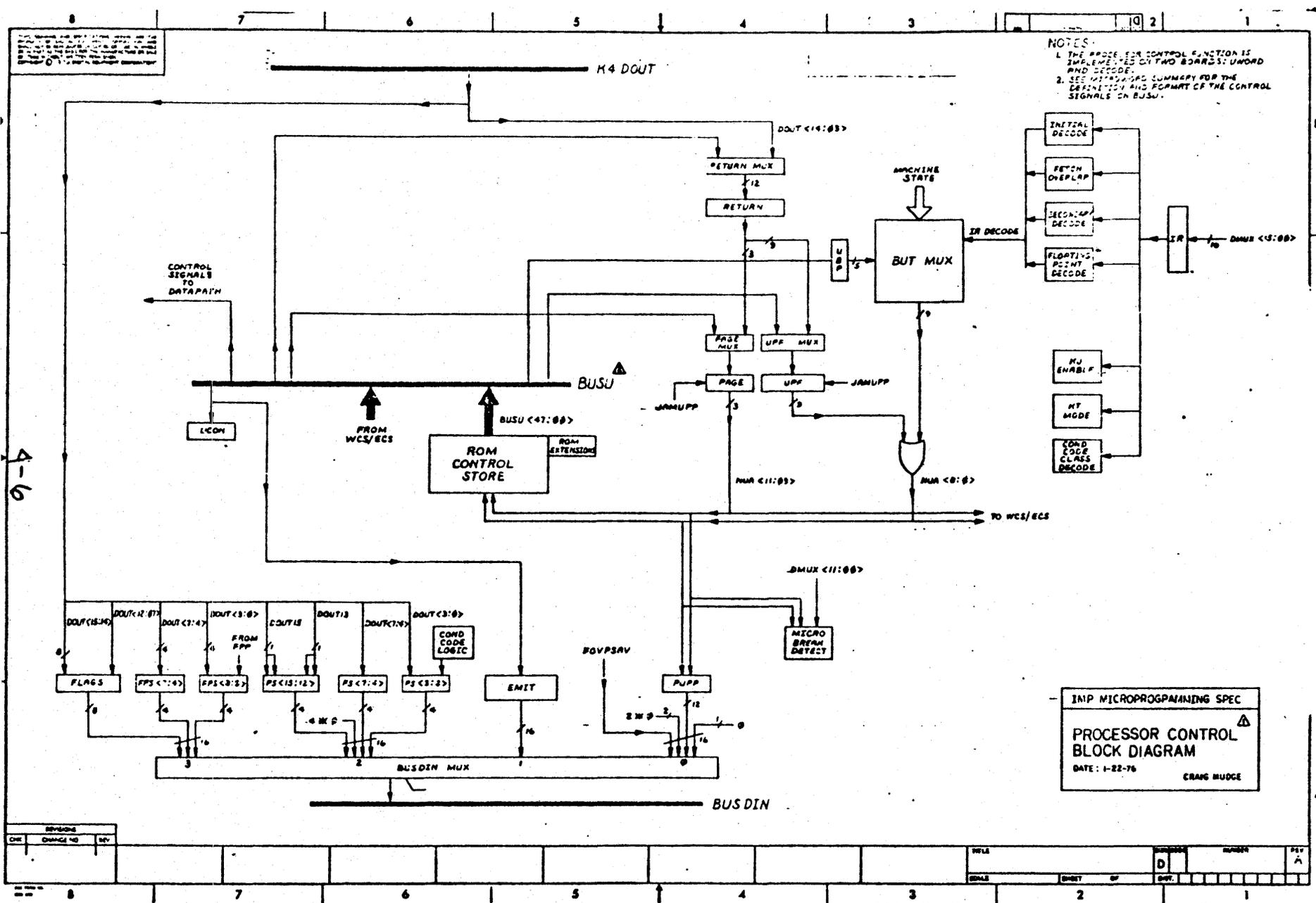
Figure 4-2

Figure 4-2 is a block diagram of the Processor Control
section of the IMP. At the top and bottom of the diagram
are the two data busses: BUS DOUT and BUS DIN. Data from the
D register can be moved into the Processor Control section
over DOUT; data from the processor control section can be
moved into the CSP over BUS DIN.Data is placed on BUS DIN by
the BUS DIN MUX.
In the middle of the diagram is BUSU, which carries the
microword signals through the processor.

4.2.1   Next Micro Address, NUA

The Next Micro Address, NUA, selects the next microinstruction
to be executed from the control store (either base machine
or WCS). NUA<11:09> are the contents of the Page register,
as shown in Figure 4-3. The Page MUX selects between the
two sources for the page register: Return<11:09> or
μ<32:30>. NUA<8:0>is the result of ORing the output
of the $\beta\mu$A register with the output of the BUT MUX. Both
the Page Mux and the $\beta\mu$A Mux select the microword input except
when BUT(RETURN) is specified.

Chapter 3 describes, from a functional viewpoint, the loading
of the Return register. You can see in Figure 4-3 that specifying
BUT(SUBR B), BUT(SUBR A), or BUT(RETURN) causes the Return
Mux to select one of its inputs; the Return Mux output
is input to the Return register.

μ<46:44,41:33>   DOUT<14:03>

RETURN MUX ← BUT(SUBRA) or BUT(RETURN)

/12

RETURN

/9

/3

μ<0:8>

μ<32:30>

Branching
Logic

BUT(RETURN) → PAGE MUX   BUA MUX ← BUT(RETURN)

JAMUPP → PAGE   BUA ← JAMUPP

/3   /9

NUA<11:09>

NUA<8:0>

to base machine c.s.   to WCS
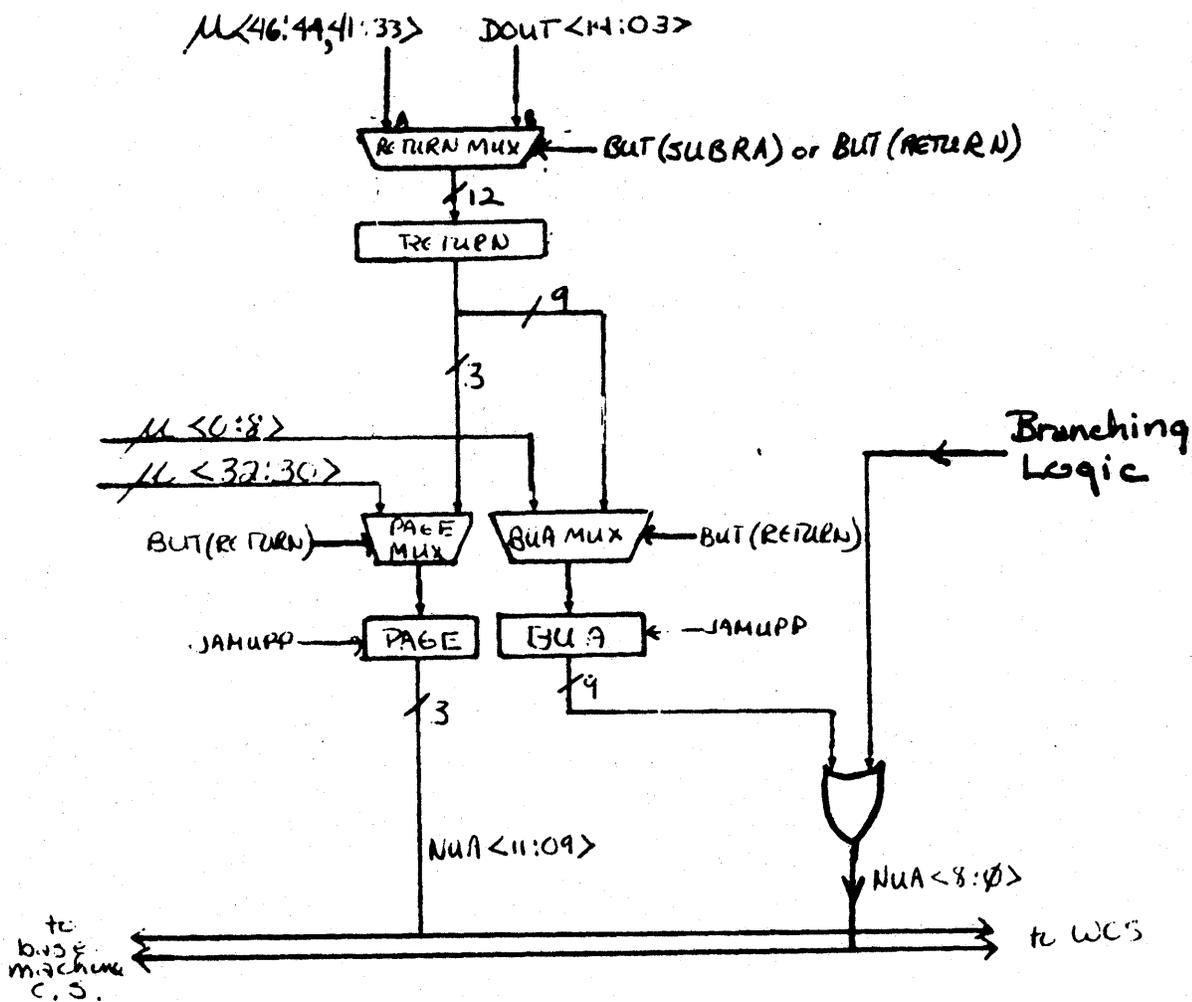
Figure 4-3: NUA Formation

Looking at the Page and BUA registers again, you will notice
a JAMUPP signal going into both of them. JAMUPP stands for
JAM MicroProgram Pointer, and the effect of this signal is
to "jam" a unique address into the NUA. This is used to dispatch
into the JAM routine, which services synchronous error conditions,
internal addresses,   etc. The JAM routine is described
in more detail in Section 5.3.

4.2.2   BUS DIN MUX

The BUS DIN multiplexer determines what data from the Processor
Control section is gated onto BUS DIN to be sent to the CSP.
This multiplexer has four inputs; selection among these
inputs is controlled by bits from the UCON register, as
shown in Figure 4-4.

Since the BUS DIN MUX is a tri-state device, it requires an
enable signal as well as selection signals. The enable signal
is also generated from a combination of bits from the UCON
register. Refer to Table 4-1.
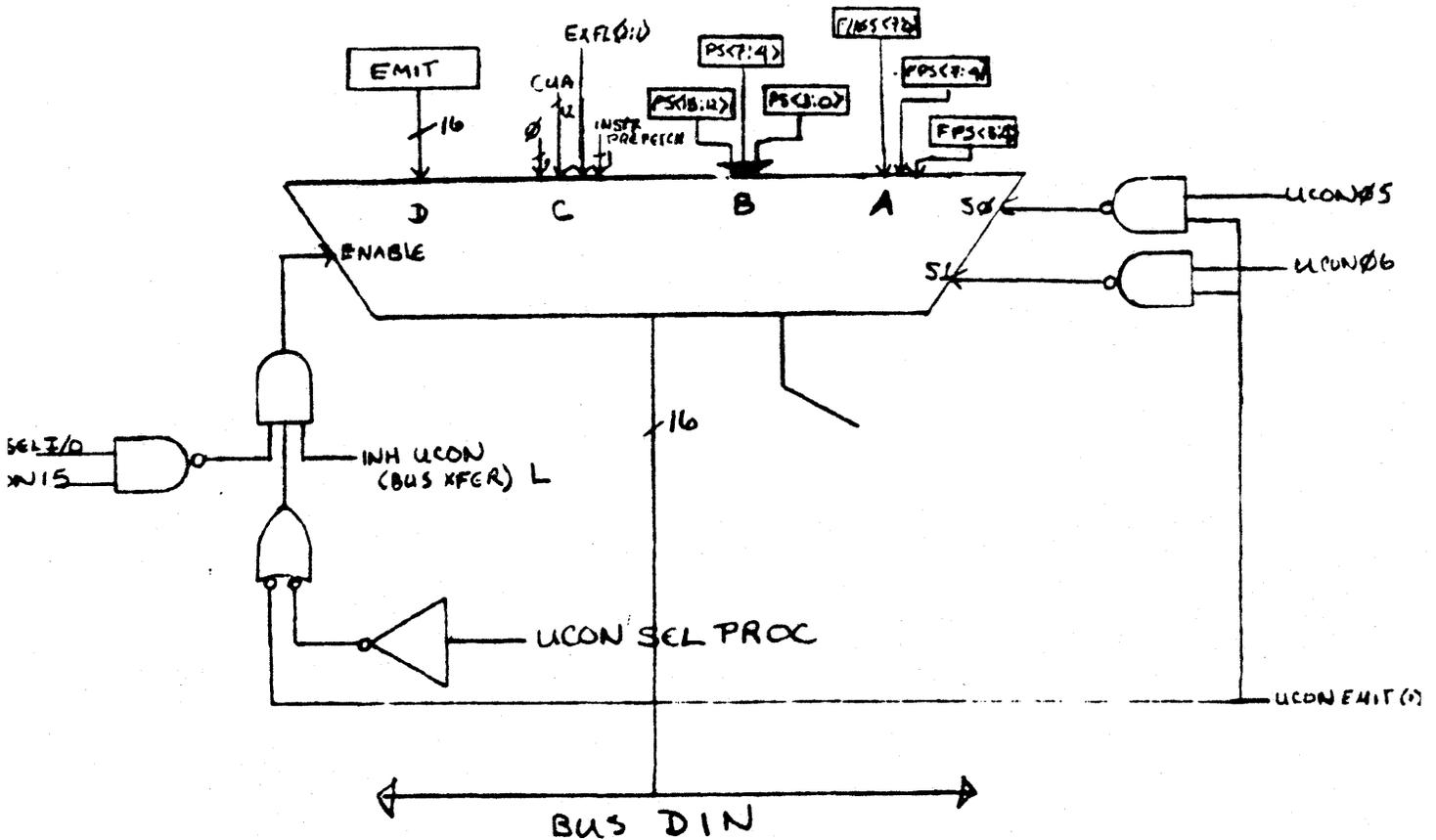
Figure 4-4   Selection and Enabling of
BUS DIN MUX

There are two ways in which the BUS DIN MUX can be enabled
and selected. The signal UCON EMIT (1) L is true whenever
the UCON register has been cleared by BUT(CLEAR FLAGS) or
INIT. UCON EMIT (1) L enables the multiplexer and forces
the selection to port D, which is the EMIT input. This
allows you to select EMIT without using a microword to set
up the UCON register.

The other method of selecting and enabling BUS DIN MUX does
require you to set up the UCON register from the microword.
The microword fields to use are KPROC, $\mu<36>$, and I/O, $\mu<46>$,
(in the UCON SELECT row of the microword summary), and KPROC
READ, $\mu<39:38>$, in the UCON READ CONTROL row of the mciroword
summary.
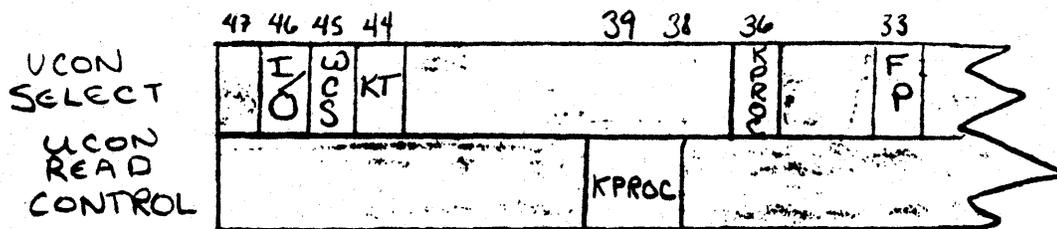
Figure 4-5   μword Fields for Controlling BUSDIN MUX

The logic for enabling the BUSDIN MUX works according to the
equation:

$$\text{ENB BUSDINMUX} = \overline{\text{UCON SEL I/O}} \wedge \overline{\text{UCONL}} \wedge (\text{UCON SEL PROC} \vee \text{UCON EMIT})$$
$$\wedge \overline{\text{INH UCON (BUS XFER)}}$$

From this, you can see that to enable the BUS DIN MUX, you must
have KPROC, $\mu<36>$ equal to 1 and I/O, $\mu<46>$, equal to 0 in
the microinstruction which loads the UCON register. The encoding

of the KPROC READ field, μ<39:38>, is shown in Table 4-1.

TABLE 4-1

KPROC READ FIELD ENCODING

KPROC READ Field Value	MNEMONIC	EFFECT
0	FLAGS, FPS	BUS DIN<15> ← FLAG<03> BUS DIN<14:11> ←FLAG<07:04> BUSDIN<10:08>← FLAG<02:00> BUSDIN<07:00>← FPS<07:00>
1	PS	BUSDIN<15:14> ← PS<15> BUSDIN<13:12> ← PS<13> BUSDIN<11:08> ← ∅ BUSDIN<07:00> ← PS<07:00>
2	CUA	BUSDIN<15> ← ∅ BUSDIN<14:03> ← CUA<11:00> BUSDIN<02:01> ← EXFLAG<2:1> BUSDIN<00> ← INSTR PREFETCH
3	EMIT	BUSDIN<15:00> ← EMIT<15:00>

The signal INH UCON (BUS XFER) L is generated by the Bus
Control section of the processor. It disables all the UCON-
controlled multiplexer on BUSDIN so that UNIBUS data can be
gated onto BUS DIN.

4.2.3 Using the IMP's Literal Facility

The EMIT field of the microword allows you to introduce a
16-bit literal into the datapath from the microword.

The contents of the EMIT field of the microword, $\mu<47:44, 41:30>$, is gated onto BUSDIN when both the S1 and S0 inputs to the BUSDIN MUX are high. EMIT is selected when the UCON register is loaded with a KPROC READ value of 3, and when BUT(CLEAR FLAGS) is issued. (BUT(CLEAR FLAGS) selects EMIT because it forces UCON EMIT (1) L to go to the low, or true state.)

Because the EMIT field overlaps the BSEL field ($\mu<41:40>$), you must use the CSPADR field to specify the address in the CSP to which you wish to write the literal data. Remember that the contents of CSP are complemented before the CSP is addressed.

As long as you issue a BUT(CLEAR FLAGS) and do not load the UCON register before the microword in which the EMIT lateral is specified, BUSDIN MUX will always be enabled onto BUSDIN, and the EMIT port will be selected.

The following example writes the number 326_8 to CSP[MD].

```
INSTR1:
    BUT(CLEAR FLAGS), J/INSTR2

INSTR2:
    EMIT/326, BEN/CSP, CSPADR/2, WRCSP/YES
```

Besides using EMIT to supply literals for datapath computation, you will find it useful for providing constants, such as those for loading the RES register. You can also load subroutine return addresses by using EMIT and then issuing BUT(SUBR A) or BUT(RETURN) in a later microinstruction.

4.2.4   Reading the Status Registers

The remaining three inputs to the BUSDIN MUX are status registers. To get data from these registers into the data-path, set up the UCON register for BUS DIN MUX enabling and selection and write a location in the CSP. If you write the data into EMITCON, CSP[6], the UCON set-up and CSP write can be done during the same microcycle.

4.2.4.1   Current MicroAddress (CUA) -- The Current Micro-Address register, CUA, tracks normal microcode flow. It is loaded with the NUA at P3. When the JAMUPP routine is invoked, CUA tracking of the microaddress is disabled, and the CUA contains the address of the microinstruction causing the JAMUPP.

The CUA is gated onto BUSDIN<14:03> when the UCON register is set up with KPROC SEL equal to one and KPROC READ equal to 2. BUSDIN<15> is loaded with zero, BUSDIN<02:01> is loaded with EXTFLAGS<2;1> , which are currently unused and reserved, and BUSDIN<00> is loaded with INSTR PREFETCH, which indicates the overlapped fetch of a macro-level instruction. As there is no macro-level instruction fetch overlap on XFC instructions, this bit should always be 0.

4.2.4.2   The Processor Status Registers -- The PDP-11 Processor Status Word (PS) is implemented on the IMP as three separate registers so that each of its parts can be written separately.

PS<15:12> are the mode bits. Because the IMP does not implement
Supervisor mode, PS<14> always has the same value as PS<15>,
and PS<12> always has the same value as PS<13>. PS<15>
indicates the current processor mode. A value of 1 indicates
the current mode is User; a 0 indicates the current mode is
Kernel. PS<13> indicates the previous processor mode; 1 for
User, and 0 for Kernel.

PS<7:4> contains the current processor priority and the
T-bit, and PS<3:0> contains the condition codes N, Z, V,
and C.

The PS is gated onto BUSDIN when the UCON register is set up
with KPROC SEL = 1 and KPROC READ = 1.

4.2.4.3 <u>Floating Point Status (Low Byte)</u> -- When the UCON
register is set up with KPROC SEL = 1 and KPROC READ equal to
0, the low byte of the Floating Point status word is gated
onto BUSDIN<07:00>. (The high byte is stored in FPSHI-FEC in
the ASP.) The format of FPSLO is shown in Figure 4-6.

7	6	5	4	3	2	1	0
FD	FL	FT	FMM	FN	FZ	FV	FC

7 Floating Double Precision Mode (FD)
   Determines the precision that is used for Floating Point calculations. When set, Double precision is assumed; when reset Floating precision is used.

6 Floating Long Integer Mode (FL)
   Active in conversion between Integer and Floating Point format. When set, the Integer format assumed is Double Precision two's complement (i.e. 31 bits + sign). When reset, the integer format is assumed to be Single Precision two's complement (i.e. 15 bits + sign).

   Floating Truncate Mode (FT)
   When set, causes the result of any arithmetic operation to be truncated. When reset, the results are rounded.

4 Floating Maintenance Mode (FMM)

3 Floating Negative (FN)
   The result of the last operation was negative.

2 Floating Zero (FZ)
   The result of the last operation was zero.

1 Floating Overflow (FV)
   The result of the last operation resulted in an arithmetic overflow.

0 Floating Carry (FC)
   The result of the last operation resulted in a carry of the most significant bit. This can only occur in integer-Floating conversions.

Figure 4-6   FPSLO

4-14

4.2.4.4　Flag Register -- The Flag register contains a
number of micro-level state indicators for the base machine.
The register contains two types of flags: short-term and
long-term. The short-term flags are cleared by BUT(CLEAR FLAGS).
The layout of the Flag Register is shown in Figure 4-7.

μBreak Enable	Service Request	(Spare)	Fast Fltpt Enabled	CSP Cnsts Invalid	MFPI	MTPI	T-bit Mask

Enables μbrk
for MED ─────────┘

To WCS on
BUT(SERVICE)─────────────┘
  (If no other service condition exists)

Fast FLTPT enabled ────────────────────────┘

If set, CSP
does not contain
Floating Point Constants ─────────────────────────┘

Masks PS(T)
for RTT

MTPI in
progress

MFPI in
progress
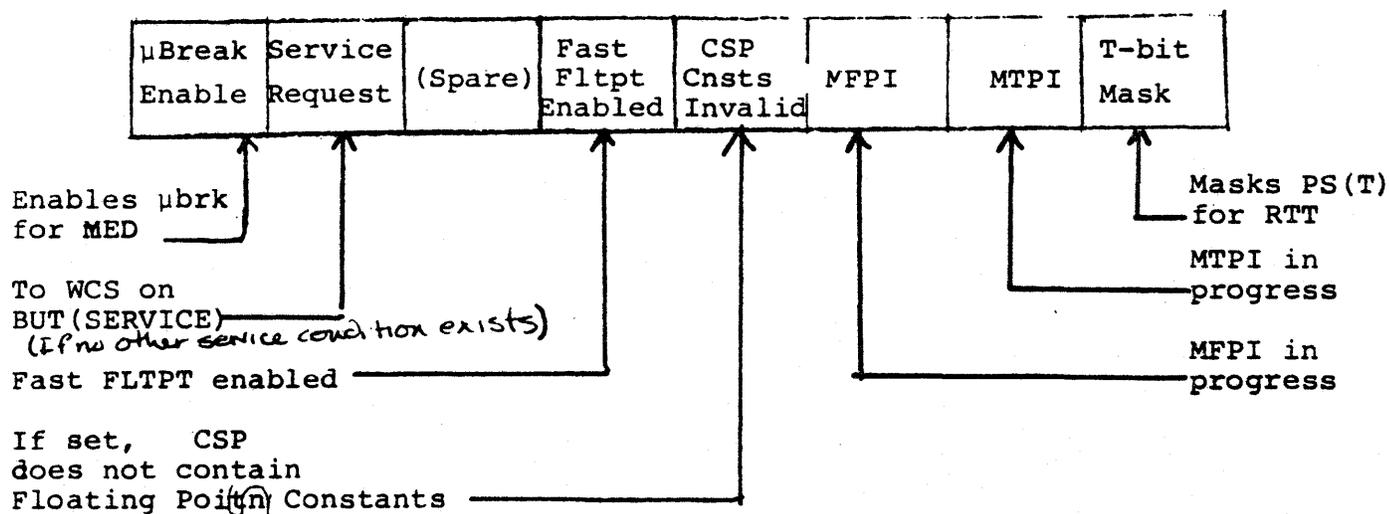
Figure 4-7　Flag Register

The CSP Constants invalid bit, FLAG<3>, is set whenever
CSP[0:13] are used to store anything other than the Floating
Point constants described in Section 2.3.

The contents of the Flag register are gated onto BUSDIN when
the UCON register is set up with KPROC SEL equal to 1 and
KPROC READ equal to 0, as shown in Table 4-1.

4.2.5   Writing the Status Registers

The registers which provide input to the Processor Control's
BUSDINMUX can be loaded from the D register. You must set
up the UCON register to indicate which registers are to be
written; set up the D register, and specify the write.

After setting up the UCON register, you specify the write by
setting the microword fields BGB, BUSBOX, and DATTB all
equal to 1.

4.2.5.1   PS ← D

The fields which set up the UCON register for writing the PS
are KPROC SEL; PS<3:0>, μ<47>; PS<7:4>, μ<34>; and PS<15:12>,
μ<31>. The three sections of the PSW may be loaded at the
same time or independently. The loading of the PS from the
D register is as follows:

$$PS<3:0> \leftarrow D<3:0>$$
$$PS<7:4> \leftarrow D<7:4>$$
$$PS<13:12> \leftarrow D<13>$$
$$PS<15:14> \leftarrow D<15>$$

With the UCON register set up, the indicated sections of the
PS are loaded from BUS DOUT when BGB, BUSBOX, and DATTB all
equal 1. The condition codes, PS<3:0>, are clocked at P2; the
other sections of the PS are clocked at P3. Thus you must set
up the D register one microcycle before you try to load the
PS , and keep D stable until P2 of the microcycle
in which the write is specified for PS<3:0> and keep D stable
until P3 of the µcycle containing a write to
other parts of the PSW,

You may prefer to set the condition codes with this method,
rather than use the CCC microword field. The logic associated
with CCC is especially designed to handle the PDP-11
instruction set. Setting the condition codes directly allows
you to have more control over the state information
transmitted to the macro-level program. *For example, you could set the V-bit, from microcode, and to use a BVS instruction at the macro level.*

For example, the following example loads the condition codes
with values previously stored in SR<3:0>.

```
        LDUCON:                    !Set up UCON prior to anything
            KPROCSEL/YES,          !else.
            PS<3:0>/YES,
            BGB/YES, BUSBOX/YES,
            CONO/YES, J/NEXT
        .
        .
        .


        SETUP:
            P2: D ← SR,            !This exapands to: AEN/XMUX,
            J/CLOCK               !XMUX/SR, ALU/SELECT A, WHEN/P2,
                                  !CLKD/YES
        CLOCK:
            BGB/YES, BUSBOX/BOX,  !Condition codes loaded at P2.
            DATTB/YES
```

Note that you do not specify any ALU activity in the micro-
instruction that sets up the UCON register. Because of the
overlapping microword fields, an ALU specification could
cause an inadvertent UCON selection.

4.2.5.2 <u>FPSLO<7:4> ← D<7:4></u> -- The four high bits of the low byte of the Floating Point Status register are loaded from D<7:4>. The UCON register must be set up with KPROC SEL and FPS<7:4>, μ<35>, both equal to 1. FSPLO<7:4> is clocked at P2 of the microcycle in which BGB, BUSBOX, and DATTB all equal 1.

Clocking of FPSLO<3:0> is controlled by an extension of the microword and cannot be performed from the WCS control store.

The high byte of the Floating Point Status word is stored in ASPHI[16].

4.2.5.3 <u>FLAG<7:0> ← D<15:08></u> -- The Flag register is loaded from D<15:08>. The UCON register must be set up with KPROC SEL and FLAGS, μ<30>, both equal to 1. The Flag register is loaded at P3 when BGB, BUSBOX, and DATTB are all equal to 1.

Remember that if you store ANYTHING in CSP [0:13], you must set the CSP invalid flag, Flag<3>.

4.3   MEMORY OPERATIONS

The 11/60 Inner Machine has three interfaces with the rest
of the system:

             DATA IN .... CSP, IR
             DATA OUT .... D register
             ADDRESS OUT ...... BA register.

The memory management unit, the cache memory, and the Unibus
are all invisible to the microprogrammer. To access a main
memory location, you must set up the appropriate registers
in the datapath and specify a Unibus cycle.

Data from the Unibus is placed on BUSDIN, which provides one
of the DMUX inputs. The other DMUX input comes from the cache.
The DMUX output goes to the CSP and to the Instruction
Register (IR), and the FP11-E.

4.3.1   The Instruction Register

The Instruction Register, IR, is in the Processor Control
section of the processor. The IR holds the first word of
a PDP-11 instruction. Control store dipatching is based on
the decoding of the contents of the IR.

Input to the IR is the same as that of the CSP: the output of
the DMUX. Obviously, not all words fetched from memory contain
PDP-11 instructions. Therefore, clocking of the IR is under
microprogram control.

There are two ways in which you can clock a word of data
into the IR. The normal method is to issue a DATA IN AND CLOCK
IR bus code, as described in Section 4.3.2. The other
method, which may be used to take advantage of the IR-based
BUTs, is to specify IR loading with the UCON register. If the
UCON register is set up with KPROC SEL and IR, μ<32>, both
equal to 1, the IR will be loaded from the DMUX during the
next microcycle in which BGB, BUSBOX, and DATTB are all equal
to 1. When IR clocking is specified, the load occurs at P2.

If you refer to the BUT list in Section 3.3.3, you will see
that there are a number of branches which test the contents
of the IR. Although these branches were designed to facilitate
decoding of PDP-11 instruction, it is possible to make more
general use of them.

For example, consider
                [Anyone have a good idea for the example
                needed here?]

4.3.2   Microword Bus Control Fields

The Bus Control Fields span μ<24:20>, as illustrated in Figure
4-8. These are the same bits that are used to control intra-
processor (UCON) communication cycles.
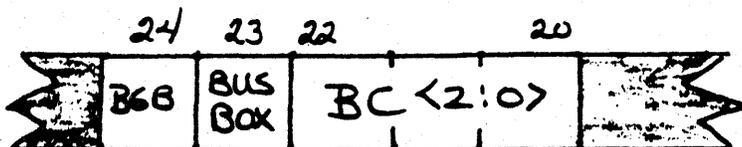
Figure 4-8 Bus Control Fields

BGB, μ<24>, must be 1, indicating that the remaining bits are to be used to control a bus cycle.

BUSBOX differentiates between a main memory (unibus) cycle and an intra-processor (UCON) cycle. A value of 0 in the BUSBOX field indicates that a main memory cycle will take place.

The BC, or Bus Control field, μ<22:20>, indicates what type of memory cycle is to take place. There are two basic types of memory cycles: DATIs and DATOs. During a ~~DATO~~ DATI, the contents of the location specified by the BA register (as relocated by the memory management unit) is gated through the DMUX, and can be clocked into the IR or the CSP. During a DATO, the data in the D register is written to the location specified by the BA register (after relocation by memory management).

Table 4-2 lists the BC codes, their mnemonics, and their functions.

DATI and DATIB cause word ~relocated~ and byte reads respectively. If the location specified by the ✓BA register is in the cache, no Unibus cycle is performed.

If the BA specified an Internal address (see Section 4.3.3) when DATI NO INT is issued, an Illegal Internal Address Access Trap will be issued.

DATIP has two functions. For core memories, it inhibits the restore cycle for locations that will be immediately written with new data. In the case of devices which can respond to more than one Unibus, the DATIP prevents the device from responding to any other requests. When a DATIP is issued, the bus will remain busy until the next bus cycle or BUT(SERVICE)

4-21

Table 4-2
BUS CONTROL CODES

VALUE	MNEMONIC	FUNCTION
0	DATI&CLKIR	Data In, IR loaded
1	DATINOINT	Data In, No internal address allowed
2	DATO	Data Out
3	DATIB	Data In, odd BA address allowed
4	DATIP	Data In, locks bus
5	DATOB	Data Out, allows odd BA address
6	DATI	Data In
7	INVALIDATE	One cache location invalidated

NOTE:
Only Byte opcodes (ADDB, etc) can issue DATIBs or DATOBs.

An **INVALIDATE** BC code does not cause a Unibus cycle. The specified location in the cache is invalidated. The next reference to that location causes a main memory reference. Subsequently, the location is again cached.

DATO and DATOB cause word and byte writes respectively. A Unibus cycle is always performed, and the cache is updated.

Note that addresses in the I/O page are never cached.

4.3.3 Internal Addresses

Some registers with Unibus addresses are not actually
connected to the Unibus, but are located within the processor
itself. These locations are called Internal Addresses.

These locations are not accessed by the Bus Control section of
the 11/60. When the contents of the BA register specifies an
Internal address, the JAM routine is invoked. The JAM routine
accesses the internal register and gates its contents
through the DMUX to the CSP. When the data is ready, control
returns to the microword which issued the bus code.

Invocation of the JAM routine by specifying an Internal Address
does alter the state of the datapath. More significantly, the
JAM routine uses the Return register, so an Internal Address
within a subroutine will cause a return to the wrong location.

Table 4-3 lists the 11/60's Internal Addresses. Notice that a
DATOB cannot be performed to some of these registers; the DATOB is
converted to a DATO.

Table 4-3

11/60 INTERNAL ADDRESSES

ADDRESS	REGISTER	DATOB CHANGED TO DATO?
772300	Kernel PDR 0	No
772302	Kernel PDR 1	No
772304	Kernel PDR 2	No
772306	Kernel PDR 3	No
772310	Kernel PDR 4	No
772312	Kernel PDR 5	No
772314	Kernel PDR 6	No
772316	Kernel PDR 7	No
772340	Kernel PAR 0	No
772342	Kernel PAR 1	No
772344	Kernel PAR 2	No
772346	Kernel PAR 3	No
772350	Kernel PAR 4	No
772352	Kernel PAR 5	No
772354	Kernel PAR 6	No
772356	Kernel PAR 7	No
777540	WCS Status Register	Yes
777542	WCS Address Register	Yes
777544	WCS Data Register	Yes
777570	Switch Register	YES

TABLE 4-3 (Cont.)

ADDRESS	REGISTER	DATOB CHANGED TO DATO?
777572	MMRØ	No
777574	MMR1	Yes
777576	MMR2	Yes
777600	User PDR 0	No
777602	User PDR 1	No
777604	User PDR 2	No
777606	User PDR 3	No
777610	User PDR 4	No
777612	User PDR 5	No
777614	User PDR 6	No
777616	User PDR 7	No
777640	User PAR 0	No
777642	User PAR 1	No
777644	User PAR 2	No
777646	User PAR 3	No
777650	User PAR 4	No
777652	User PAR 5	No
777654	User PAR 6	NO
777656	User PAR 7	No
777744	Memory System Error Reg.	Yes
777746	Cache Control Register	Yes
777752	Hit Miss Register	Yes

Table 4-3 (Cont.)

ADDRESS	REGISTER	DATOB CHANGED TO DATO ?
777766	CPU Error Register	Yes
777770	Ubreak Register	No
777774	Stack Limit Register	Yes
777776	Processor Status Word	No

4.3.4   Timing Considerations

Data from memory is introduced into the datapath through

the CSP. The loading signal for the CSP occurs at P3 if

WRCSP is specified in the microword.

The cache and memory management logic take a finite amount
of time to process a request. Thus, even when the requested
data is in the cache, there is not enough time between P1
(when the BA is loaded) and P3 (when CSP data must be valid)
for the data to be gated through the DMUX. The loading signal
for the CSP must be delayed until the next microinstruction.
This situation is illustrated in Figure 4-9.
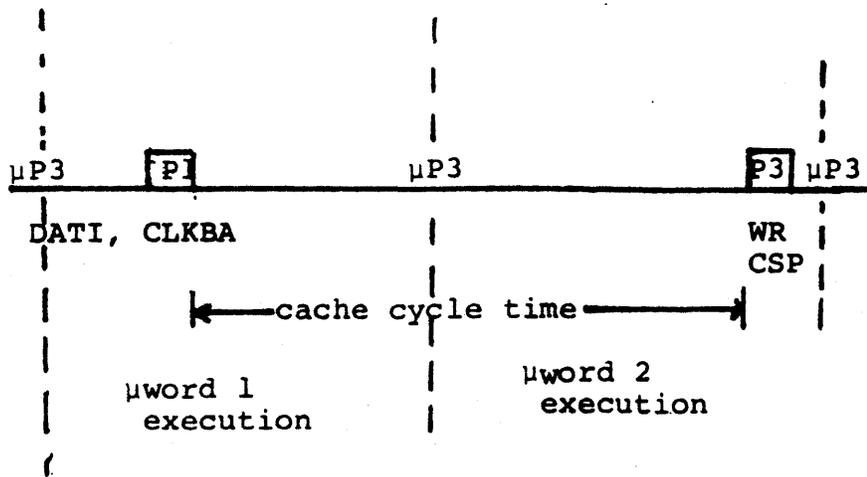
Figure 4-9   DATI Timing

Every DATO to a valid Unibus address involves a Unibus cycle
as well as a cache cycle. The cache update, with the address
specified by the memory management unit and the data specified
by the D register, begins at P3 of the first microcycle. The
Unibus cycle does not begin until after P2 of the second
microcycle. Hence, the data to be written must be clocked into
D during the first microcycle, and kept constant until P3
of the cycle following the DATO (i.e., until the Unibus cycle
is complete). The procedure for doing DATOs to valid Unibus
addresses is shown in Figure 4-10.
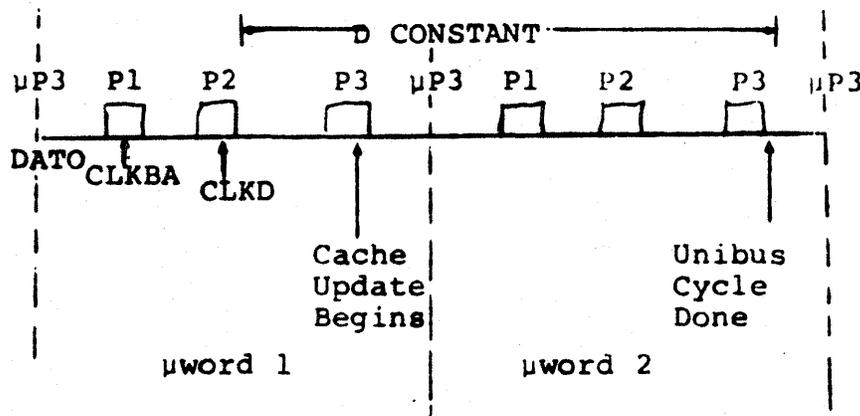
Figure 4-10   DATI Timing

Because both DATIs and DATOs require two microcycles to
complete, do not specify memory references in two successive
microwords. The invocation of the Unibus in two consecutive
microcycles will put the machine in an undefined state.

4.3.5   Examples

                T.B.S.

4.4   THE CACHE/KT SECTION

The Cache/KT section of the processor contains the memory
management logic, the cache, the stack limit, and the DMUX.
Virtual addresses from the BA register are relocated by the
memory management logic, and the resultant Physical Bus
Address is directed to the Unibus and to the Cache. A hit
(data in cahce) on a DATI causes the Cache port of the DMUX
to be selected; on a miss, the Bus Control section places the
data on BUS DIN and it is written into the cache from the
DMUX output. The NPR address monitor invalidates cache locations
which have been altered during DMA transfers. The stack limit
unit compares the stack address with a previously loaded value,
and causes an error if the stack goes below the stack limit.

4.4.1   The Cache

The 11/60 cache memory consists of 1024 words of direct-mapping
cache. Each word consists of a tag field and a data field. The
tag field has seven address identification bits, a valid bit,
and byte parity. The data field consists of two eight-bit bytes,
each with a parity bit.

Each location in backing store can be directly mapped, or
allocated, to one specific cache slot and each cache slot can
accept data from up to 128 different backing store locations.

The Cache Control Register, CCR, is used to modify cache operation
for diagnostic purposes. CCR$\langle 6 \rangle$ is used to write wrong parity.
When set, it causes opposite (Odd) parity to be generated in the
tag and data fields. When read, those locationswill cause a parity

UNIBUS

DOUT

NPR
ADDRS
MONITOR

MEMORY
MANAGEMENT

(KT)

BA

STACK
LIMIT
(KJ)

PBA

PBA

CACHE

INT.
ADDR
ROM

CCR

SLR

NMR2

MMR2

CACHE
VALID

HIT

DMUX

R/W REG MUX

RO REG MUX

UCON09

PAR

PDR

MMR0

TRAP

UCON12
UCON KTSEL

UCON
05

UCON06

UCON10
UCON KTSEL

UCON11
UCON KTSEL

BUS DIN

4-30
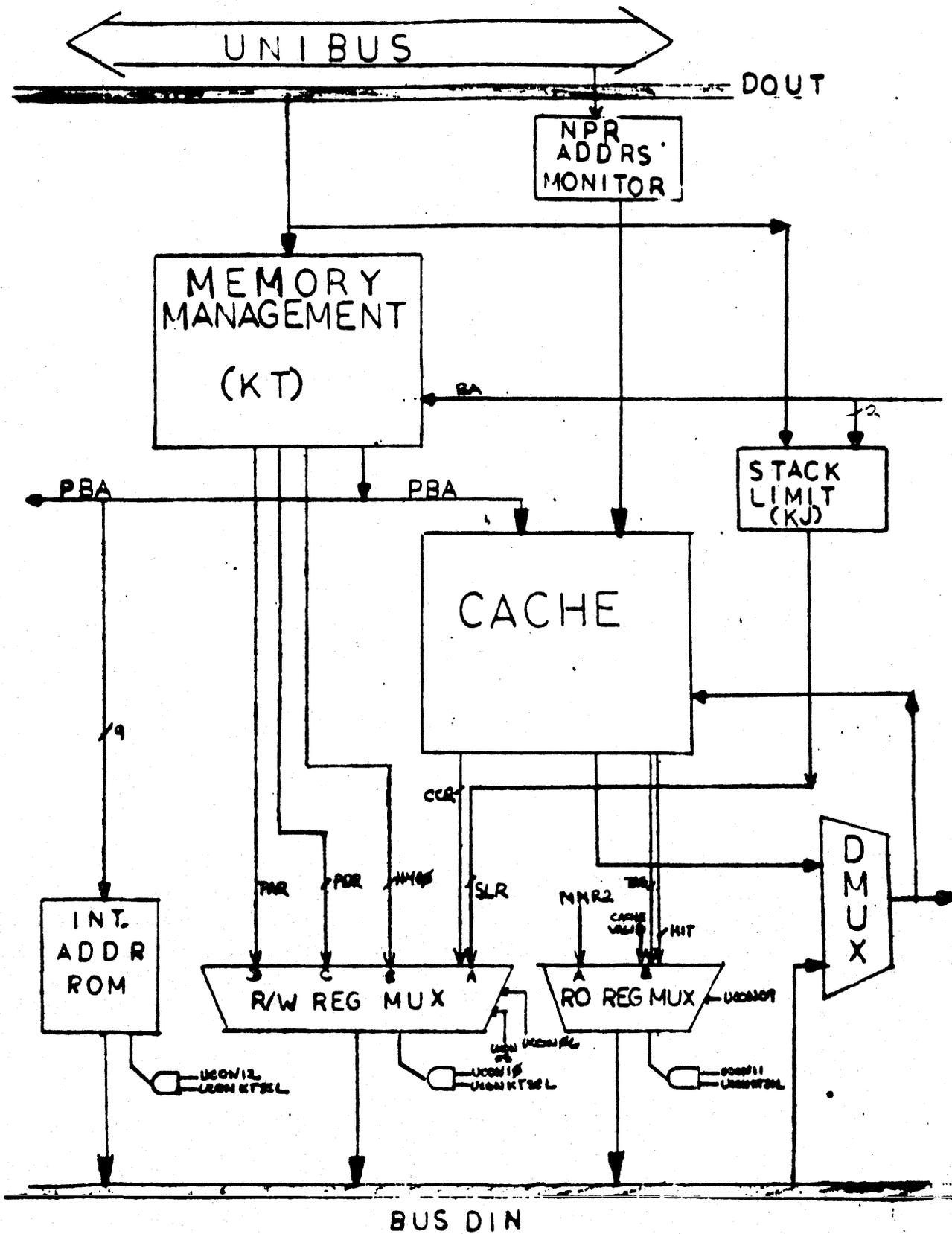
CACHE CONTROL REGISTER (777746)

FORCE MISS 0
FORCE MISS 1
WRITE WRONG PAR
BAD CPU JAM

HIT/TAG (777752)

TAG FIELD   HIT REGISTER
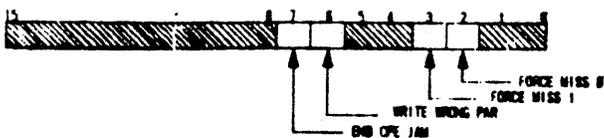
STACK LIMIT REGISTER (777774)

STACK LIMIT

MMR0 (777572)

ENABLE RELOCATE
PAGE NUMBER
MODE OF OPERATION
MAINTENANCE MODE
ABORT-READ ONLY OPERATION
ABORT-PAGE LENGTH ERROR
ABORT-NON-RESIDENT
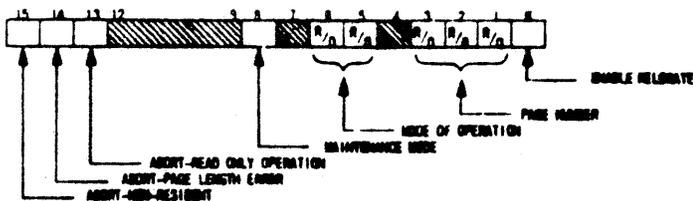
MMR2 (777576) · READ ONLY

16 BIT VIRTUAL ADDRESS

(772340 - 772356) KERNEL
PAGE ADDRESS REGISTERS (777640 - 777656) USER

PAR FIELD
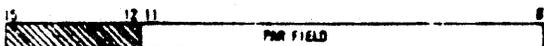
(772300 - 772316) KERNEL
PAGE DESCRIPTOR REGISTERS (777600 - 777616) USER

PAGE LENGTH FIELD   ED   ACF
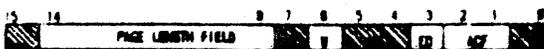
NOTE
1 MMR1(777574) READS AS ALL 0'S
2 MMR3(772516) NOT USED
3 16BM R/0 = READ ONLY BIT

4-31

error and inhibit the hit signal. CCR⟨7⟩ is CPE JAMUPP, which, if
set, causes an access to be aborted if a cache parity error occurs.
CCR⟨3:2⟩ are used to force misses. CCR⟨2⟩ will causes misses
whenever PBA⟨10⟩ is 0 (cache locations 0 -511). CCR⟨3⟩ will
cause misses whenever PBA ⟨10⟩ is 1 (locations 512- 1023).

The Hit/Miss register indicates whether the six most recent references
by the CPU were hits or misses. A 1 indicates a hit, and a 0
a miss. The most recent cycle is tracked in the low-order bit. This re
register is read-only.

4.4.2   Accessing KT/Cache Registers

These cache registers, along with the memory management registers
and the stack limit register, can be accessed from the datapath
over the UCON interface. Some of the registers are read-only
at the microcode level, as they are at the macro-code level.

The KT/Cache section has three devices which can gate data onto
BUS DIN: the internal address ROM, the Read-Write multiplexer,
and the Read-only Register multiplexer. The enable signals for
these devices are provided by UCON data bits: multiple enables can
cause hardware damage. Thus you must be very careful to properly
set up the UCON register, and not attempt ALU operations during
the same cycle as a UCON set-up.

The particular PAR or PDR gated onto BUS DIN is determined by the
current processor mode (User or Kernel) and, within those sets,
by BA⟨15:13⟩

KT/CACHE UCON INTERFACE

UCON XFER	KT SEL	14	13	12	11	10	9	8	7	6	5	FUNCTION	
	1	1	0	0	0	0	x	x	x	x	x	INHIBIT RELOCATE	
	1	0	1	0	0	0	x	x	x	x	x	ENABLE DISPATCH FOR INT ADR	
	1	0	1	1	0	0	x	x	x	x	x	BUS DIN ← INT ADR <15:00>	
	1	0	0	0	1	0	0	x	x	x	x	BUS DIN ← MMR2 <15:00>	READ REGISTERS
	1	0	0	0	1	0	1	x	x	x	x	BUSDIN<15>←CACHE VALID BUS DIN<14:08>←CACHE ADDR<17:11>L BUS DIN <05:00>←HIT<5:0>	
	1	0	0	0	0	1	x	0	0	0	0	BUS DIN <15:00>← SLR<15:08>, CCR<7:0>	
	1	0	0	0	0	1	x	0	0	0	1	BUSDIN ← MMR0	
	1	0	0	0	0	1	x	0	0	1	0	BUSDIN<14:08,06,03:01>← PDR	
	1	0	0	0	0	1	x	0	0	1	1	BUSDIN ← PAR	
1	1	0	0	0	0	0	x	0	1	0	0	CCR<7,6,3,2>← Dout<7,6,3,2>	
1	1	0	0	0	0	0	x	0	1	0	1	MMR0<00>←Dout<00>	
1	1	0	0	0	0	0	x	0	1	1	0	PDR<03:01>←Dout<03:01>	
1	1	0	0	0	0	0	x	0	1	1	1	PAR<7:0>←Dout<7:0>	WRITE REGISTERS
1	1	0	0	0	0	0	x	1	0	0	0	SLR<15:08>←Dout<15:08>	
1	1	0	0	0	0	0	x	1	0	0	1	MMR0<15,13,08>←Dout<15:08>	
1	1	0	0	0	0	0	x	1	0	1	0	PDR<14:08>←Dout<14:08>	
1	1	0	0	0	0	0	x	1	0	1	1	PAR<11:08>←Dout<11:08>	
1	1	0	0	0	0	0	x	1	1	0	0	CCR, SLR ← DOUT	
1	1	0	0	0	0	0	x	1	1	0	1	MMR0<15:13,08,00>←Dout<15:13,08,00>	
1	1	0	0	0	0	0	x	1	1	1	0	PDR<14:08,03:01>←Dout<14:08,03:01>	
1	1	0	0	0	0	0	x	1	1	1	1	PAR<11:00>←Dout<11:00>	

4.5   THE BUS CONTROL SECTION

The Bus Control section of the 11/60 has four main functions:

    Unibus interfacing and arbitration
    Console interfacing
    Timing control
    Status control

This section has three interfaces to BUS DIN, as shown in
Figure 4-xx. The DU (Data-Unibus) register buffers Unibus data
on DATIs. The BUS DIN multiplexer gates console data, service
flags, error information, and the physical bus address onto
BUS DIN. The Data Storage (DS) register allows data from DOUT
to be gated onto BUS DIN. This is used for cache updating on
DATOs, and for writing data into the CSP and IR.

Nearly all of the activities on this board are transparent to
the user microprogrammer. Furthermore, meddling with this
logic offers the greatest potential for putting the CPU into an
undefined state. Thus the following sections do not suggest
using the facilities of this section. It is described here for
informational purposes.

Note that the signal IHN UCON (BUS XFER), which must be false
for other UCON activities to take place, is generated in this
section. When either the DU or DS registers has data to be gated
onto BUS DIN as a result of a DATI or DATO, it takes precedence
over all other BUS DIN devices. Thus, the signal IHN UCON (BUS XFER)
is gnerated by the hardware when either of these conditions is

detected.
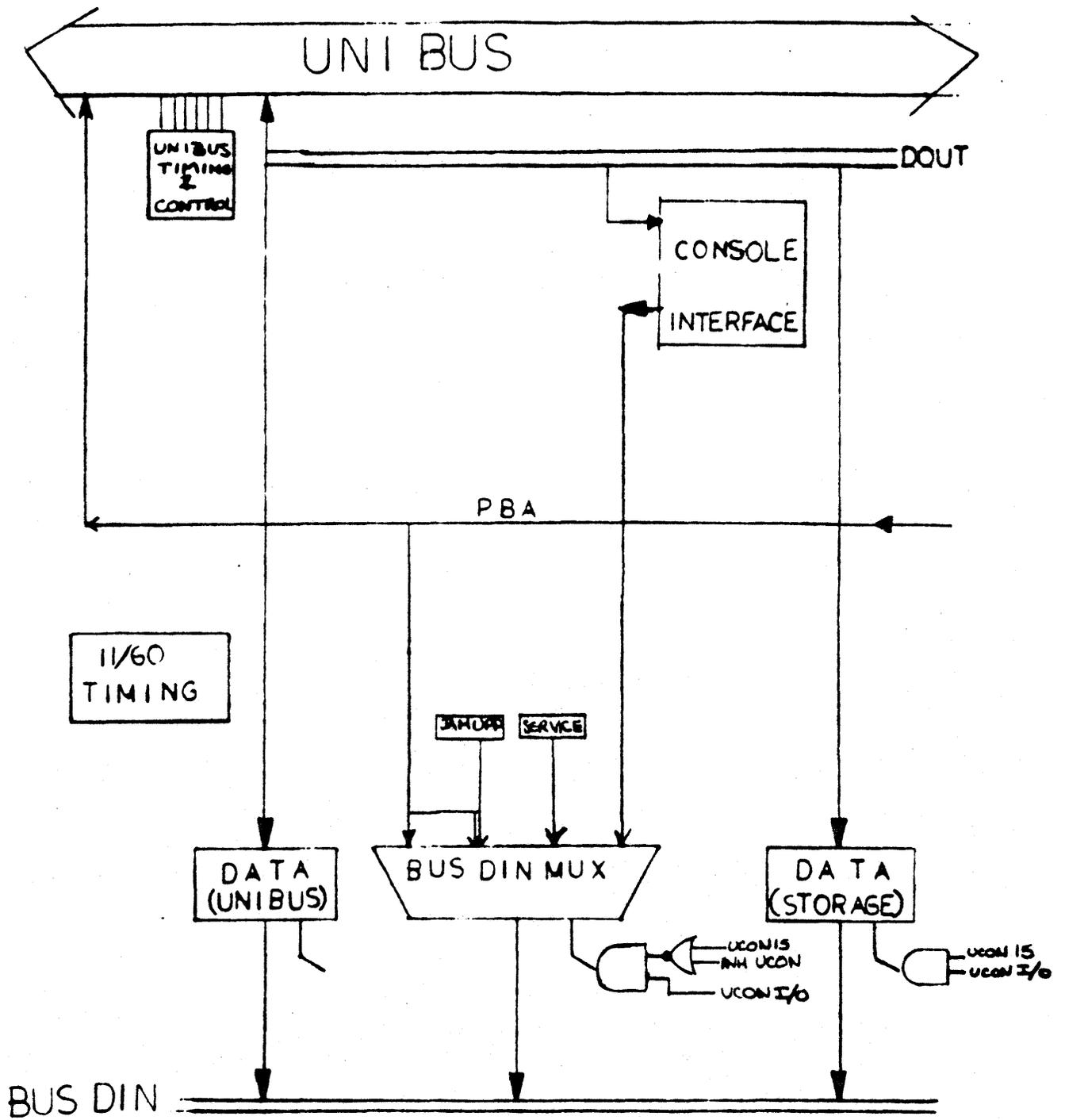
Microprogram control of the Unibus is essentially limited to
issuing bus codes and checking for Unibus requests (BRs)
via BUT(SERVICE) or BUT(BG).

4.5.1   The PDP-11/60 Console

The PDP-11/60 operator's console is shown in Figure 4- .
There are five discrete visual displays which indicate
the current operation of the processor. These lights and
their meanings are as follows.

    RUN -- when lit, indicates CPU is running code
    PROC -- when lit, indicates CPU is Unibus master
    USER -- when lit, KT-11D is in User mode
    CONSOLE -- when lit, indicates processor is in console mode
    BATTERY -- if on steadily, battery backup is present and
                charged. Slow flashing indicates battery is
                charging; rapid flashing indicates battery is
                discharging. If off, battery is not present or dead.

The numeric display register contains six octal charachters.
It can display data or addresses. When displaying addresses,
all decimal points are lit.

They key switch has five positions. The panel lock position
deactivates all keypad functions, and inadvertent operation
of the slide switch has no effect..
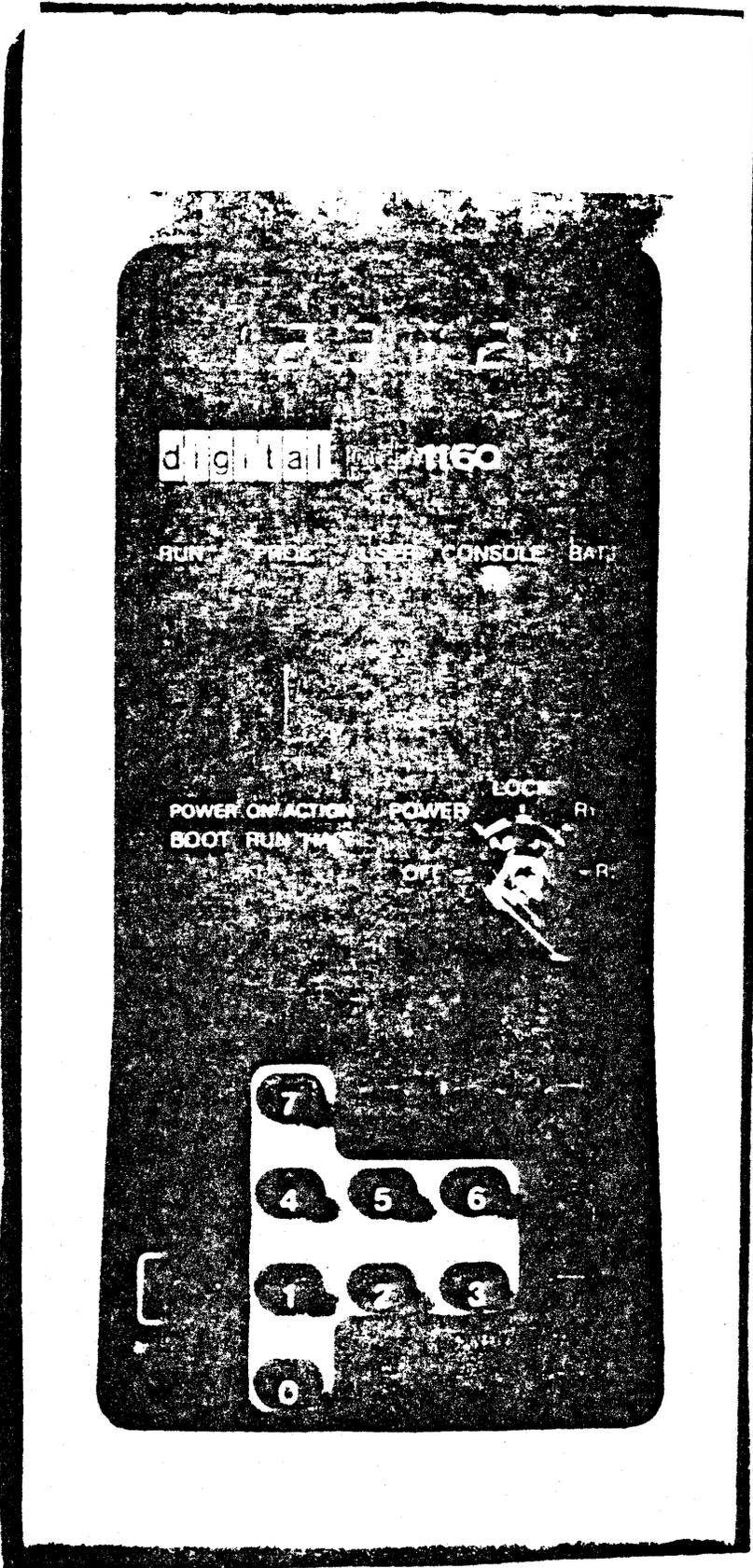
The three-position slide switch allows a choice of action to be taken on power-up. If the switch is in the HALT position, the CPU will power-up in console mode. If the switch is in the RUN position, power-up will trap to location 24 (power-fail vector). If the battery backup on MOS memory has failed, the M9301 bootstrap will be invoked, which is the action taken if the switch is in the BOOT position.

The hardware for control of the operator's console is in the bus control section of the processor.(M7877). Keypad entris are encoded on the console board (KY-11P) and trasnmitted to the status board by means of a 40-wire cable. This interface is shown in Figure 4- .

Keypad entries are read twice and compared: the five-bit key code is directed to the BUS DIN MUX. If the keycode is valid and the comparison showed both readings equal, the console service request flag is set.

Console microcode is entered from the service flow when a service request is detected and no higher-priority service condition exists.

4.5.2 Console Datapath Registers -- The following datapath registers are reserved for use by the console microcode:

```
        CNSL.TMPSW := BSPHI[7]
        CNSL. CNTL := ASPHI[3]
        CNSL. SW   := ASPHI[6]
        CNSL. ADR  := ASPHI[7]
```
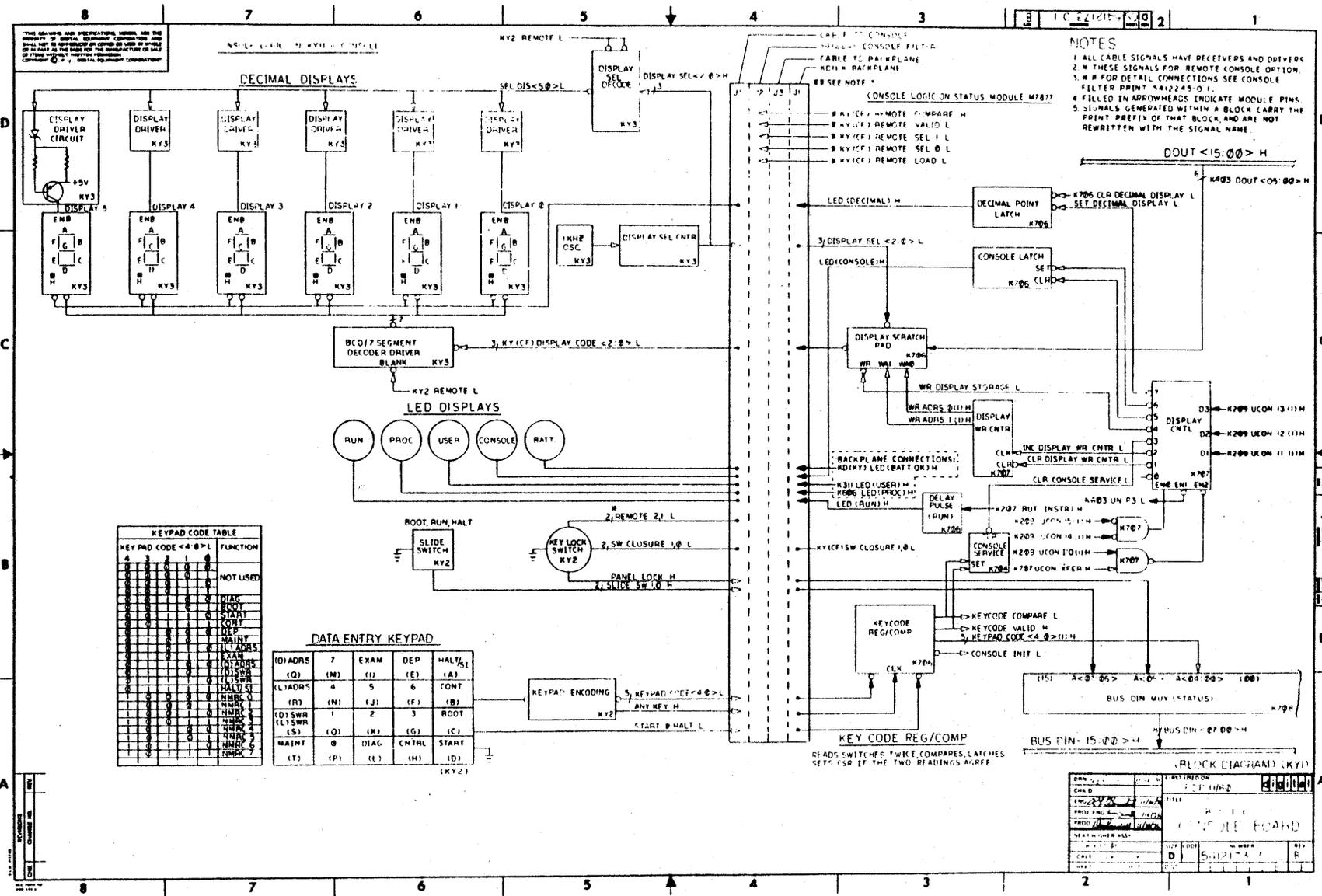
(BLOCK DIAGRAM) (KY)

CONSOLE BOARD

The Temporary Switch register, CNSL.TMPSW, is used to hold
the value displayed in the octal display on the console. When
any numeric key is pressed, its binary value is placed in
the low-order three bits of the temporary switch register, with
the previous contents shifted left three bits. Program
movements to the Switch register are disabled, during log process.

The Console Control register, CNSL.CNTL, contains control and
status bits, as well as CNSL.SW<17:16>, CNSL.TMPSW<17:16>,
and CNSL.ADR<17:16>. The layout of this register is shown in
Figure 4- .

The Switch Register, CNSL.SW, has Unibus address 777570. It
is loaded with the contents of CNSL.TMPSW when the Load
Address and Control keys are simultaneously pressed. It can be
accessed by a macro-level program; however, if the Display
Lock (DISLOCK) bit in CNSL.CNTL is set, the move will be
treated as a no-op.

CNSL.ADR, the console address register, is loaded from the
temporary switch register when Control and Load Address are
pressed simultaneously.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CONSOLE	DLOCK	EXAM	DEP	SWREG		TSR		SI	ONTC DSCR					CONSOLE ADDRESS	
				17	16	17	16							17	16
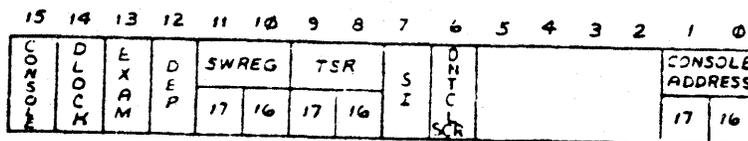
R (CNTL) IN B SP.

Figure 4-11

4-41

4.5.3   Console Microcode

The BUT(SERVICE) at the end of every macro-level instruction dispatches
to the console microcode if the console service resquest flag
is set. The entry point is CSR01. The console microcode loads
console constants into the CSP and sets the CSP Invalid flag
(see Section 2.   ). This is done by calling two subroutines.
FLG reads the Flag register from the Processor Control section
and places it in R(TEMP1) . FLGS ORs a constant from MD with
R(TEMP1) and re-writes the Flag register. The 'EXAM', 'DEP',
and 'DON'T CLR CSR' bits in CNL.CNTL are then cleared. The
console tests for single-step mode and halts if SI is set.

If single-stepping is not indicated, the A-port of the BUS DIN
multiplexer (in Bus Control) is selected, and the data is
read into MD. Microcode branches decode the keypad code
and dispatch to the appropriate console service routine, as
shown in the console flow diagram, Figure 4-   .

4.5.4   Console Use of UCON Interface

Data to be displayed on the console is moved from CNSL. TMPSW
onto DOUT, and then written into the Display Scratchpad. The
display scratchpad is continously read (sequentially) to
drive the octal display.

Control for the dipaly Scratchpad, the console mode indicator,
the decimal displays, and for clearing the console service
request flag comes from the Display Control decoder. This
decoder uses UCON 13:11 as its data inputs and is enabled
when BEGIN, UCON, and XFER $\mu \langle 24,23,21 \rangle$ are all equal to 1, and
UCON $\langle 15 \rangle$ and UCON $\langle 14 \rangle$ are both equal to 0. Table 4-   shows
the console UCON codes and their functions.

CONSOLE ENTRY

LOAD SCR CONSTANTS
CLEAR EXAM, DEP, 'DNTCLRCSR'

IS 'SI' SET ? — YES / NO

CLEAR EXAM.DEP

DISPLAY TSR

CLEAR TSR

SI ← ∅

HLT ∅1

CMND COMPLETE

CLEAR CMMD LATCH
SET 'D LOCK'
READ SWITCH CODE
BRANCH ON SWITCH CODE

UNUSED CODES

ALL OTHER CODES (EXCEPT THE UNUSED CODES)

LSWR

DSWR

NUMERIC

CMND REJECT

IS 'CONSOLE' SIT ? — NO / YES

CMND ILLEGAL

SWREG GETS TSR

TSR GETS SWREG

TSR <17:00> GETS ('TSR<14:20> + NUME ∅0<2.0>)

SI = 1   SI = ∅   SI = ?

'DNTCLRCSR'

BRANCH

DISPLAY TSR

'CSR' ← ∅

IS 'CONSOLE' SET ? — NO / YES

BOOT

START

CONT

DEP

LADRS

DADRS

HALT/SI

IS 'SCR' SET ? — NO / YES

CONSOLE ← ∅

'CONSOLE' ← ∅
'CSR' ← ∅
TSR ← ∅

'CONSOLE' ← ∅
'CSR' ← ∅
TSR ← ∅

DEP ROUTINE

'EXAM' ← ∅
'DEP' ← ∅
CAR ← TSR

'DLOCK' ← ∅
TSR ← CONSOLE ADDRESS
DP ← 1

IS 'CONSOLE' SET ? — YES / NO

SI

BOOT

PC ← R(CADRS)

CMND COMPLETE

HALT

DIAG

CMND COMPLETE

FET ∅1

MAINT

EXAM

SI ← 1

CONSOLE ← 1

SELECT 'WCS' HOT BOX

CMND COMPLETE

DBUF LATCH GETS R(SW)
IR ← ∅

EXAM ROUTINE
TSR ← DATA

'CONSOLE' ← ∅

'DP' ← 1

BRA∅9

FET ∅1

TSR ← ∅

TSR ← PC

BG ASSERTED — N / Y

INTR ROUTINE

MED ∅2

FET ∅1

FE1 ∅1

R (CNTL) IN B SP.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CONSOLE	DLOCK	EXAM	DEP	SWREG 17 16		TSR 17 16		SI	DNTCLRCSR					CONSOLE ADDRESS 17 16	

11/6C MICROPROGRAMMING SPEC

CONSOLE FLOW DIAGRAM

DATE: 3-29-76

4-43

Table 4-
Console UCON codes

	UCON SET-UP		ENABLE	FUNCTION
I/O SEL	<15:14>	<13:11>	BEGIN,UCON,XFER	
1	00	001	YES	CLR Display WR Counter
1	00	010	YES	INC Display WR Counter
1	00	011	YES	CLR Console Service Rqst
1	00	100	YES	WR Display Storage
1	00	101	YES	CLR Console LED
1	00	110	YES	SET console LED
1	00	111	YES	Set Decimal Display

4.5.5 Bus Control BUSDIN Mux

Console data is moved into the datapath through the BUS DIN
multiplexer. This multiplexer is enabled onto BUS DIN when the
UCON register is set up with UCON<15>=0 and UCON I/O SEL = 1
and the INH(UCON(BUS XFER) signal is not asserted. Slection
of the multiplexer is done by UCON<10:9>. The data placed on
BUS DIN by each of the selection codes is shown in Table 4- .

BUS DIN MUX FOR STATUS

BUS DIN	UCON <10:9> 0	1	2	3
15			ODD ADRS ERR(1) H	PBA 15 H
14		BS SERVICE (1) H		PBA 14 H
13			SSYN TIMEOUT (1) H	PBA 13 H
12		NPR TIMEOUT (1) H	YELLOW ZONE (1) H	PBA 12 H
11		DATOB (1) H	RED ZONE (1) H	PBA 11 H
10		DATO (1) H	WCS PAR ERR H	PBA 10 H
09		I/O PAGE PA 17 H	KX(PWR) DISCHARGED L	PBA 09 H
08		I/O PAGE PA 16 H	MEM PAR ERR (1) H	PBA 08 H
07	SLIDE SW 1 L	HIBYTE PAR ERR L	SSYN TIMEOUT (1) H	PBA 07 H
06	SLIDE SW 0 L	LOBYTE PAR ERR L	CACHE ERR (1) H	PBA 06 H
05	PANEL LOCK L	TAG PAR ERR L	ILLEGAL ADRS (1) H	C ? (1) H
04	KEYPAD CODE 4 (1) H	FLTPT SERVICE H	MM ABORT (1) H	BA 04 (1) H
03	KEYPAD CODE 3 (1) H	CONSOLE SERVICE (1) H	RED ZONE (1) H	BA 03 (1) H
02	KEYPAD CODE 2 (1) H	PWR FAIL (1) H	ODD ADRS ERR (1) H	BA 02 (1) H
01	KEYPAD CODE 1 (1) H	CACHE ERR (1) H	WCS PAR ERR H	BA 01 (1) H
00	KEYPAD CODE 0 (1) H	YELLOW ZONE (1) H	U BREAK (1) H	BA 00 (1) H

4.5.6   The DS Register

The primary function of the DS register is to provide a latch
for DOUT data for writing DATO data into the cache. However,
since DIN is gated into the DMUX in the Cache/KT section of
the processor, DS also provides a path from D to the CSP and the
IR.   A DATO Unibus cycle is not required to enable DS onto BUSDIN.

The DS register can be loaded (at p3) from DOUT by setting up
UCON with I/O SEL and UCON <15> both equal to 1. The DS register
is cl OCked at P3 of the microinstruction in which UCON XFER
(BEGIN UCON XFER) is specified.

4.5.6.1 -- The DMUX   The DIN port of the DMUX (see Figure 4-)

Figure 4-   DMUX

is the default selection. Although the DMUX is in another section
of the processor, UCON<08> is used in conjuction with UCON I/O SEL
to select the cahce port of the multiplexer. This is used only
when there is a Cache hit on a DATI.

BUS CONTROL UCON INTERFACE

UCON XFER*	UCON I/OSEL	15	14	13	12	11	10	9	8	7	6	5	CLOCK	FUNCTION
1	1	0	0	0	0	1	x	x	x	x	x	x	P3	"CLEAR" DISPLAY WR CNTR
1	1	0	0	0	1	0	x	x	x	x	x	x	P3	INC DISPLAY WR CNTR
1	1	0	0	0	1	1	x	x	x	x	x	x	P3	CLR CONSOLE SERVICE FLAG
1	1	0	0	1	0	0	x	x	x	x	x	x	P3	WR DISPLAY STORAGE
1	1	0	0	1	0	1	x	x	x	x	x	x	P3	CLR CONSOLE LED
1	1	0	0	1	1	0	x	x	x	x	x	x	P3	SET CONSOLE LED
1	1	0	0	1	1	1	x	x	x	x	x	x	P3	SET DECIMAL
	1	x	x	x	x	x	x	x	x	0	0	1	P2	CLR POWER FAIL MODE
1	1	x	x	x	x	x	x	x	x	0	1	0	P2	CLR JAM ERRORS
1	1	x	x	x	x	x	x	x	x	0	1	1	P2	CLR NPR TIMEOUT
1	1	x	x	x	x	x	x	x	x	1	0	0	P2	CLR POWER FAIL FLAG
1	1	x	x	x	x	x	x	x	x	1	0	1	P2	CLR YELLOW ZONE
1	1	x	x	x	x	x	x	x	x	1	1	0	P2	
1	1	x	x	x	x	x	x	x	x	1	1	1	P2	BUS RESET (UCON)
∅	1	0	x	x	x	x	0	0	x	x	x	x	NA	BUSDIN<15:08>←∅,<07:06>←<SLSW,05> Panel Lock
∅	1	0	x	x	x	x	0	1	x	x	x	x	NA	BUSDIN ← STATUS INFO ><04:00>← Key Code
∅	1	0	x	x	x	x	1	0	x	x	x	x	NA	BUSDIN ← JAMUPD INFO
∅	1	0	x	x	x	x	1	1	x	x	x	x	NA	BUSDIN<15:00> ← PBA<15:00>
1	1	1	x	x	x	x	x	x	x	x	x	x	P3	DS ← DOUT<15:00>
∅	1	1	x	x	x	x	x	x	x	x	x	x	NA	BUS DIN ← DS<15:00>
∅	1	x	x	x	x	x	x	x	1	x	x	x	NA	SELECT CACHE DATA PORT OF DMUX

* UCON XFER = BEGIN ∧ DATTB ∧ CONO

4.5.7 Other BUS Control UCON

The remaining UCON data bits are used to clear various error and service flags. These functions are included in Table 4-   , which summarizes the UCON interface to the Bus Control section.

4.6   The WCS Section

The Writable Control Store section of the IMP can function
either as a 1K-by-48 control store or as a high-speed
3K-by-16 local store. Most users will configure the store
to be part control store and part local store.

The store is loaded 16 bits at a time from DOUT, under control
of the UCON interface. When the memory is read, data goes
out on BUSU<47:00> if the WCS is in control store (CS) mode,
or on BUSDIN<15:00> if in local store (LS) mode.

When in LS mode, the WCS is cycling as a data store and so is
not available as a source of control signals for BUSU. An
auxiliary source for BUS U signals is provided by the TMS
(Transfer Micro Store) ROM. The TMS ROM is a 512-by-16
store which controls datapath activity while the WCS is
acting as a data store.

This section describes the organization of the WCS option;
the user interface is described in later chapters. The distinction
between organization and use can be shown by a discussion of
the loading mechanism.

DOUT provides the path for passing a 16-bit word and its
associated 12-bit address to the WCS section. The UCON inter-
face is used to select and then set-up the WCS to receive
the address and data. Once the WCS section has been set up,
its local control takes over. This local control, implemented
by CROM (Control ROM) , effects such actions as clocking the
Address Register, selecting the Address Mux, and generating a
write pulse for the array.
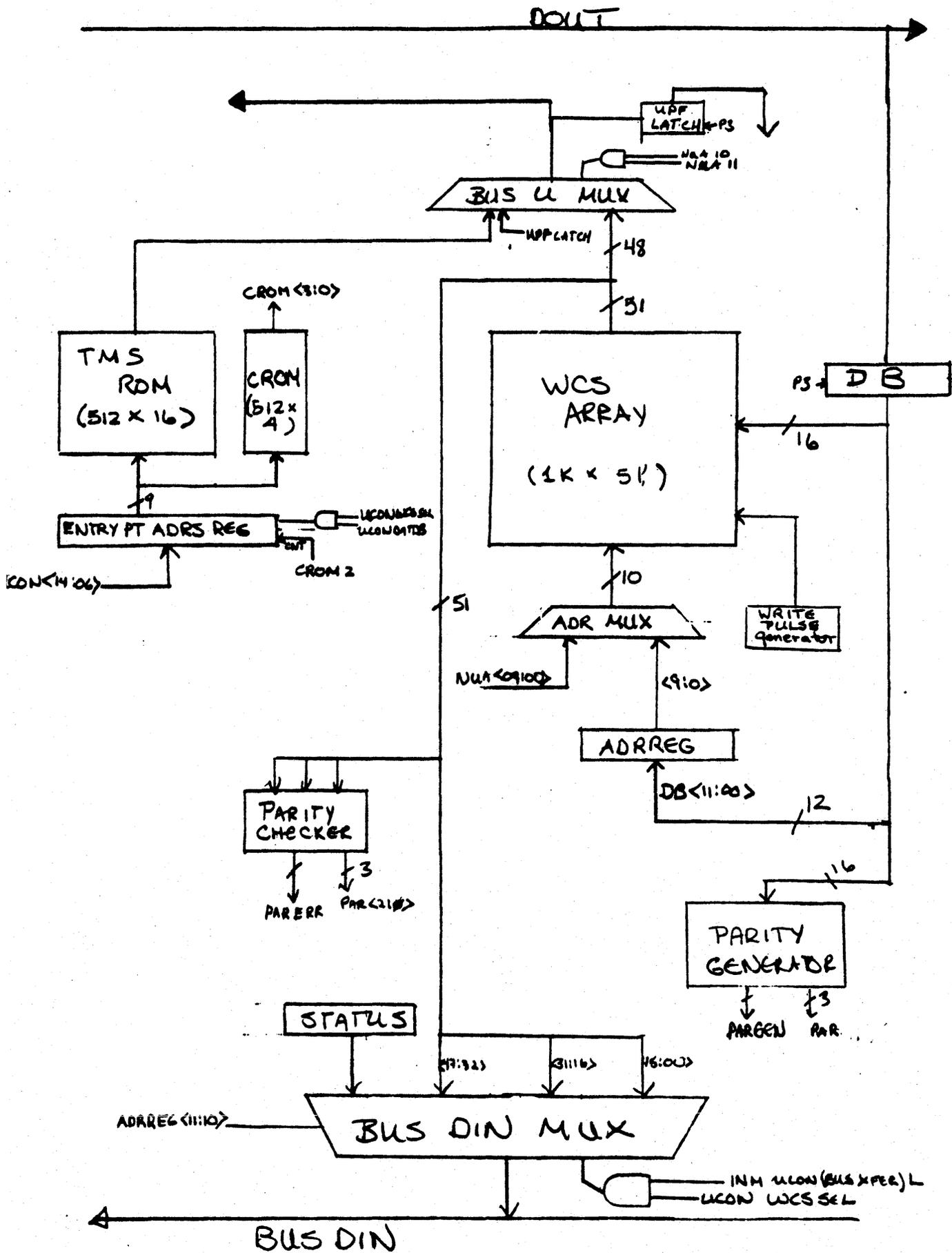
Figure 4-14   WCS Block Diagram   4-49

However, you will not interface to the WCS section at this
level. Using a macro-level program, you will move data into
the WCSDR and WCSAR registers in the PDP-11 I/O page. An
instruction which addresses these register, e.g.,
MOV #501, WCSAR, is executed by the base machine using the
primitives described in the preceding paragraph.

A block diagram of the WCS section is shown in Figure 4-14.

4.6.1   Addressing Structure of the Array

The WCS array is divided into three sections, as shown in
Figure 4-15. Each section is 1024 words long and 16 bits wide.
The array is addressed by the output of the ADRMUX.
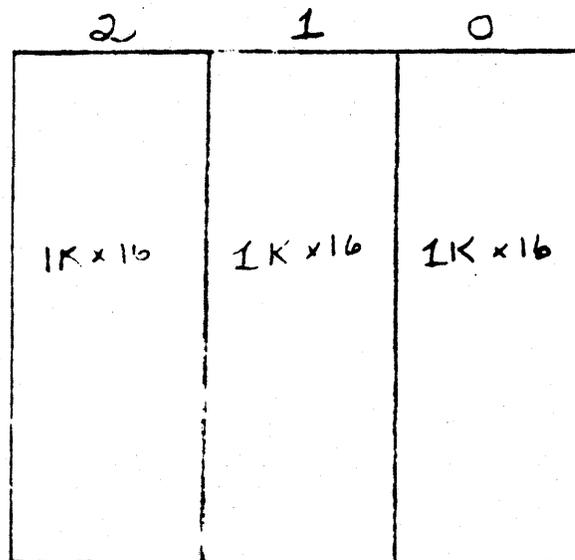
Figure 4-15

When the WCS is in local store mode, each section, or column,
is linearly addressed 0-1023. ADRREG<11:10> provides the column
address, and ADRMUX<9:0>provides the row address. See
Figure 4-16.

When the WCS is in control store mode, ADRMUX<11:10> is always
equal to 1 1₂. The entire row (48 bits) indicated by ADRMUX<9:0>
is put on BUS U.

CS mode

11 10	9 0
1 1	NUA <9:0>

LS mode

11 10	9 0
col	row

FROM ADRREG

From ADRMUX
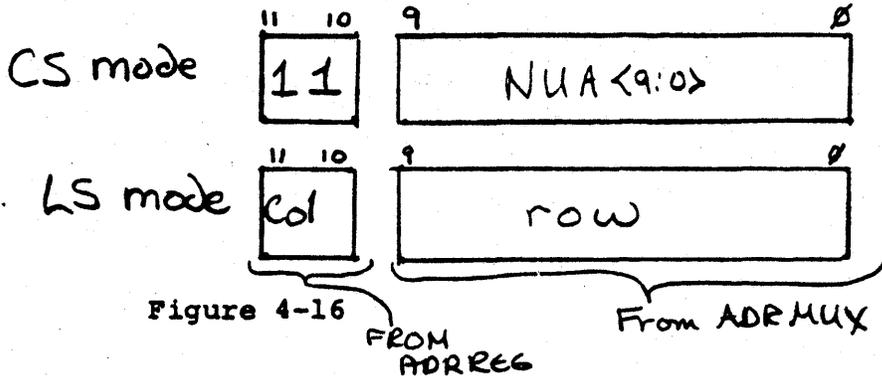
Figure 4-16

A feasible user configuration of the WCS array is shown in
Figure 4-17. There are three section of local store, A, B,
and C, each linearly addressed from 511 to 1023. Page 6 of
the 11/60 control store address space is allocated for
WCS control store.

Section C ADR<11:10>=10	Section B ADR<11:10>=01	Section A ADR<11:10>=00	1023
LS	LS	LS	ADRMUX<9:0>
			511

511

NUA<8:0>

CS
NUA<11:9>=6

0
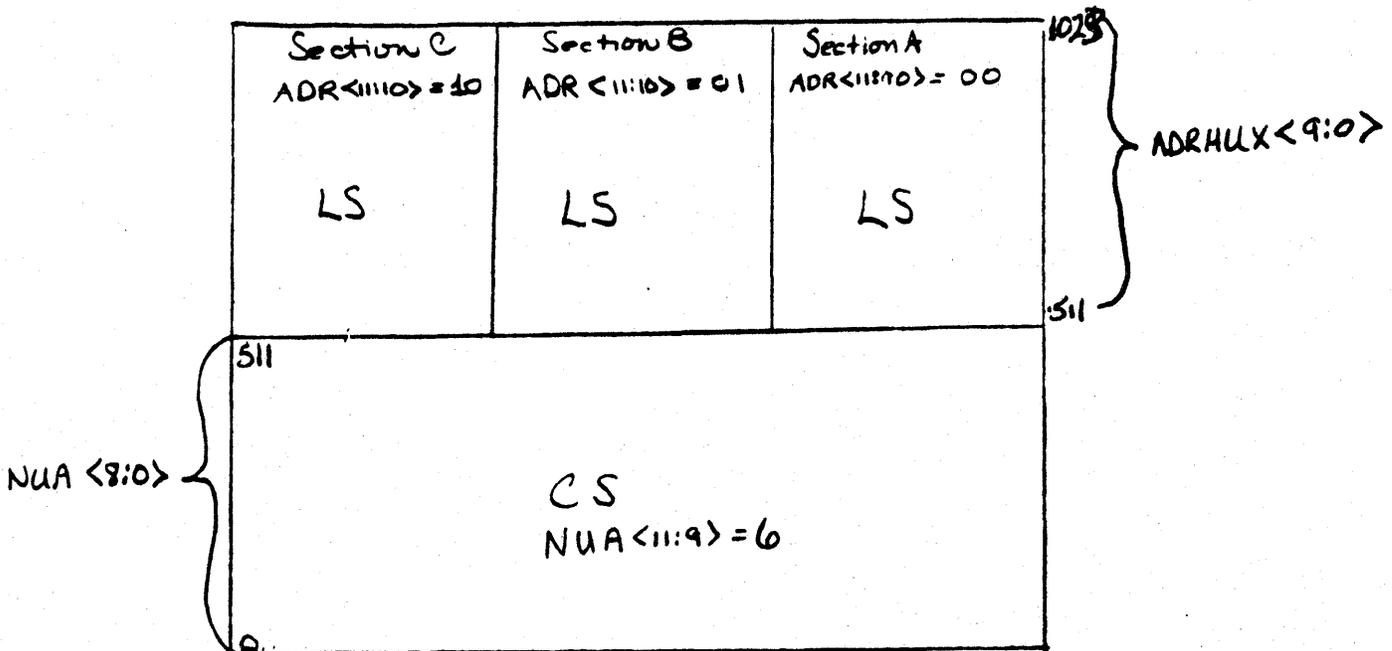
Figure 4-17   Typical User Configuration

4.6.2   Transfer of Control

NOW ENTRY POINT ADDRESS REGISTER

The TMS Pointer register, TMSPTR, addresses both the TMS ROM
and the CROM. The TMS Pointer is loaded with UCON<14:06>,
which defines the starting address of a TMS routine. In
subsequent cycles, TMSPTR is incremented if CROM<2> is a 1.

The TMS pointer is loaded when WCS SEL , μ<45>, BGB, BUSBOX,
DATTB, and CONO are all equal to 1. The TMSPTR value is
specified by the bits loaded into the UCON register. For
example, the following field value specifications would
load the TMSPTR with 010:

```
        TMSPTR ← 010   := WCS/1, BGB/1, BUSBOX/1, DATTB/1, CONO/1,
                          UCONH/0, UCONM/0, UCON10/0, UCONL/20
```

4.6.3   DB Register

The DB register stores the contents of DOUT so that 16 bits
can be written into the WCS Array during each microcycle. It
is clocked at P3. When the WCS is set up, the first word clocked
into DB is the starting array address. Subsequently, the
DB register gets the data to be written, while the Array
Address register selects the array addresses. The data in DB
is written into the array only if CROM<3> is asserted.

4.6.4   Array Address Register

The Array Address register (ADRREG) is initially loaded with
DB<11:00> , defining the starting address in the array. In
subsequent cycles, this register acts as a counter, incrementing
if CROM<1> is a 1.

4.6.5 Array Address Mux

The Array Address Multiplexer (ADRMUX) selects between the output of the ADRREG and the NUA signals from the Processor Control section. When the WCS is being loaded, or used as a local store, this multiplexer selects the ADRREG output to address the WCS array. When the WCS is in control store mode, the array is addressed by the NUA in the same way as the base machine control store is.

4.6.6 The WCS Array

The WCS Array is a 1K-by-51 RAM. Each 16-bit section has a parity bit associated with it. Even parity is generated. Only 16 bits of the array can be written at one time. When functioning as a control store, 48 bits are read onto BUS U<47:00>.

4.6.7 BUS U MUX

The BUS U multiplexer selects between the two sources of control located in the WCS section: the TMS ROM and the WCS RAM. It is a tri-state mux, and is enabled by NUA<10> and NUA<11>, which indicate the top 1K of address space is being accessed. Selection is controlled by CROM<2> both equal to 1.

4-53

4.6.8   BUS DIN MUX

When 16-bit words are read from the WCS array (LS mode), the
BUS DIN multiplexer selects which 16 bits of the 48 are
put on BUS DIN. In this situation, the multiplexer selection
is controlled by ADRREG<11:10>. If a status cycle is underway,
this multiplexer puts status information on BUS DIN.

4.6.9   Control Rom

BIT	FUNCTION
0	Address Register load enable
1	Address Register Count Enable
2	Array Address Mux select, Entry Point count enable
3	Write Pulse enable

4.7   USING WCS AS LOCAL STORE

While the WCS array is being used as a local store, the TMS
Rom provides the control signals for the datapath. Routines
in the TMS ROM provide control for loading WCS locations
from any of the scratchpad register or for loading a set of
scratchpad registers from locations in the WCS array.

To use this facility, you issue a Local Store Function Code (LSFN)
over the UCON interface, passing a local store address in
the D register as a parameter.

Each LSFN maps directly to a starting address of a TMS routine.

```
 1
 2
 3      !        TMS ROM MICROCODE FOR 11/60
 4      !
 5      !        THIS MICROCODE GOES INTO THE TMS ROM (TRANSFER MICROSTORE
 6      !        ROM).   THIS ROM RESIDES ON THE WCS BOARD AND ALLOWS A PROGRAM
 7      !        RUNNING IN THE WRITEABLE CONTROL STORE OF THE 11/60
 8      !        TO USE PART OF THIS SAME CONTROL STORE AS A BLOCK DATA STORE.
 9      !        (LOCAL STORE)   THIS ABILITY IS REALIZED BY ROUTINES WHICH
10      !        PERFORM BLOCK LOADS AND STORES OF VARIOUS PARTS OF THE INTERNA
11      !        STATE OF THE 11/60.   THE FOLLOWING PORTIONS OF THE MACHINE
12      !        ARE LOADED OR STORED:
13      !
14      !                (1) GENERAL REGISTERS
15      !                (2) WARM FLOATING POINT REGISTERS
16      !                (3) C SCRATCHPAD EXCEPT BASE CONSTANTS
17      !                (4) USER SCRATCH REGISTERS
18      !                (5) ENTIRE A SCRATCHPAD
19      !                (6) ENTIRE B SCRATCHPAD
20      !                (7) ENTIRE C SCRATCHPAD
21      !
22      !        THIS MICROCODE ALSO HANDLES ALL WCS SUPPORT NEEDED BY THE
23      !        BASE MACHINE TO PERFORM ITS FUNCTIONS.   THE FOLLOWING IS
24      !        A LIST OF THESE ENTRY POINTS AND THEIR FUNCTIONS:
25
26      !
27      !        TMS ADDRESS       FUNCTION
28      !
29      !        0001              USED BY WCSINIT FLOW.   USED TO SET ADDRESS
30      !                          REGISTER TO ZERO AND ALSO WRITES ZERO IN
31      !                          THE WORD.
32      !                          IF USED BY WCS CODE THEN LOADS ADDRESS REGISTE
33      !                          WRITES ADDRESS VALUE INTO THAT ADDRESS AND INC
34      !                          THE ADDRESS REGISTER BY ONE.
35      !
36      !        0004              USED BY WCSINIT FLOW.   WRITES A COUNT INTO WC
37      !                          THEN INCREMENTS THE ADDRESS REGISTER.
38      !
39      !        0010              LOADS WCS ADDRESS REGISTER WITH VALUE AND THEN
40      !                          DATA INTO THIS ADDRESS.
41      !
42      !        0020              LOADS WCS ADDRESS REGISTER WITH VALUE.
43      !                          (BASE MACHINE ALSO SAVES THIS SAME VALUE IN TH
44      !                          SCRATCHPAD 21).   THIS ROUTINE ALSO PUMPS ONTO
45      !                          THE DATA FROM THIS LOCATION.
46      !
47      !        0030              USED BY FIRST WORD IN ROUTINE THAT READS WCS S
48      !                          NOTE THAT THE WCS STATUS IS NOW READ BY THE UC
49      !                          INTERFACE.   THIS WORD CAN PROBABLY BE REMOVED
50      !                          BASE MACHINE AND THIS ROUTINE FROM THE TMS ROM
51      !
52      !        0040              NOT REFERENCED BY THE BASE MACHINE.
53      !
54      !        THE WCS USER CAN ALSO USE THESE ROUTINES IN THE TMS ROM.
```

.TOC   TMS MICROCODE

USING ROUTINES IN THE TMS ROM.

THE ROUTINES IN THE TMS ROM ARE DESIGNED TO SAVE DIFFERENT
SETS OF THE 11/60 MACHINE STATE INTO WCS ACTING AS A LOCAL STORE
AND ALSO TO RESTORE THESE SETS FROM DATA IN THE LOCAL STORE.

THESE ROUTINES ARE DESIGNED FOR OPTIMUM DATA FLOW TO FACILITATE
IMPLEMENTATION OF FUNCTIONS SUCH AS CONTEXT SWITCHING WHICH MUST
HAPPEN AS FAST AS POSSIBLE.   BECAUSE OF THIS OTHER USES OF THESE
ROUTINES AND SUBSETS OF THESE ROUTINES MAY NOT BE AS EASY TO USE AS
WOULD BE LIKED.

ALL ROUTINES ARE ENTERED WITH THE WCS LOCAL STORE MINUS ONE (LSADR-1)
CLOCKED INTO D.   RETURN TO THE WCS ROUTINE WILL OCCUR AFTER THE
FUNCTION HAS BEEN COMPLETED.   THESE ROUTINES ARE IMPLEMENTED
BY SETTING UP A PIPELINE IN THE DATAPATH WHERE TWO DIFFERENT
PARTS OF THE DATAPATH MOVE DURING THE SAME MICROCYCLE.   THE PIPELINE
CONTINUES UNTIL ALL DATA IN THIS SET HAS BEEN MOVED.

USING SUBSETS OF THESE ROUTINES TO MOVE ONLY A FEW OF THE DATA
ITEMS AND NOT THE WHOLE SET IS NOT EASY.   AS AN EXAMPLE THE
FOLLOWING IS THE PROCEDURE TO SAVE REGISTERS R3-R0:

(1)   USE A ROUTINE TO LOAD THE ADDRESS-2 INTO THE ADDRESS REGISTER.
(2)   CLOCK R3 INTO THE D REGISTER.
(3)   SET THE TMSPTR WITH ADDRESS THAT WRITES R4 INTO THE
         ARRAY AND MOVES R3 THROUGH THE DATAPATH AND CLOCKS IT
         INTO D. ONLY THE CROM BITS ON THIS INSTRUCTION WILL BE
         EXECUTED.   THE TMS BITS WILL NOT BE ACCESSED.   THIS WILL
         WRITE R3 INTO THE LOCAL STORE ADDRESS-1.
(4)   THE NEXT INSTRUCTION WILL WRITE R3 INTO THE ADDRESS AND
         MOVE R2 INTO D.   THE REST OF THE ROUTINE WILL WRITE
         R2-R0 INTO THE ARRAY AND RETURN CONTROL TO THE WCS
         ROUTINE AT THE THIRD INSTRUCTION AFTER THE ONE THAT
         SET THE TMSPTR VALUE.

THIS EXAMPLE SHOWS THAT A SUBSET OF THE DATA ITEMS
CANNOT BE STORED IN THE SAME MANNER AS THE ENTIRE SET SINCE

To illustrate how to invoke a TMS routine, the following
example loads the nine 16-bit words in LS[j], ... LS[j+8]
into ASPLO[0:5, 16,6:7] and BSPLO[0:5,16,6:7].

SETUP1:
        TMS PTR ← LOADGRS, J/SETUP2   ! UCON set-up
                                             ! for loading GRs
SETUP2:
               D ← j minus 1, J/NEXT

4.8   <u>UCON CONVETIONS</u>

I. Don't do a UCON Select in the cycle following a BUT(CLEAR FLAGS).

II. Keep EMIT on BUSDIN most or all of the time -its a real
time-saver.

III. Watch out for accidentally enabling multiple UCONs by
trying to perform an ALU-related function in the same word
as a UCON setup. Dedicate a microword to enabling and loading
the UCON register.

Figure 4-59

CHAPTER 5
MICROPROGRAM INTERFACES

The preceding chapters have focu ssed on the aspects of the
11/60's hardware most visible to the user microprgrammer.
However, the 11/60's architecture is not completely defined
by specifying tis hardware organization because it is a
highly microprogrammed machine.

The microcode architecture is important to the WCS user for
the following reasons:

1. It determines the environment that exists
   upon entry to the WCS

2. It expects certain state conditions to exist
   after the completion of WCS control

3. The user can cause base machine code to be invoked
   without intentionally exiting from the WCS

4. The base machine code has capabilities not
   available to WCS code

5. The base machine code provides a large set of
   examples, both of hardware usage and of
   microprogramming the 11/60.

This discussion is also motivated by the fact that no
description of a microprogrammedmachine is complete without
some discussion of the microcode.

5.1 FLOW OF THE BASE MACHINE CODE

The overall structure of the base machine code is shown in
Figure 5-1. The instruction fetch uses two microinstructions,
FET01 and FET02. FET01 is the primary entry point to which
control must be returned. FET03 issues a BUT(INSTR1), a
brach which performs initial instruction decode to approximately
75 targets. Any necessary source and destination calculations
are then made, and the instruction is executed. A test for
service is made, using BUT(SERVICE). If no service condition
exists, control is returned to the fetch sequence.

5.1.1 Overlapped Fetch

In certain circumstances, the PDP-11/60 performs an overlapped
macro-level fetch. Register-to register operations, for
example, only require one microcycle to complete, so the
overhaed of FET01 and FET02 are eliminated by fetching the
next instruction while the register-to-register instruction is
being executed. Figure 5-2 indicates the logical flow of the
overlapped and non-overlapped fetch. Hard-wired logic detects
those instructions which cannot be overlapped, and inhibits
the overlapped fetch in FET03.

5-2

E

SER Ø -FOVP   SER Ø FOVP   -SER Ø FOVP   -SER Ø -FOVP

FET Ø1 | BA ← PC, PC ← PC + 2
         DATI & CLKIR & NOT INTERNAL

SERØ2 SERVICE
a. ASYNCHRONOUS ERRORS
b. NON-FATAL ERRORS
c. INTERRUPTS
d. ....
e. USER, U-LEVEL SCHEDULED REQUEST
f. DEFAULT

SERØ1 | PC ← PC - 2

DECODE,
BA ← PC,
[IF FOV] DATI & NOT INTERNAL
[IF FOVP] & CLKIR,
           D ← PC + 2,
[IF INH]   FL ← D

FET Ø2 | IR ← DATA
         MD ← DATA
         CLEAR FLAGS

JAMØ1 JAM LPP
a. SYNCHRONOUS ERRORS (NOT YELLOW ZONE)
b. POWER UP
c. INTERNAL ADDRESS
d. OTHERS EFFECTING JAMFRC
e. DEFAULT

[TABLE A]   INSTR 1 BRANCH

SM≠Ø   DM≠Ø   R-R   R   BRANCH   JUMP   MOV #SMØ   CS*   FLT. PT.   OTHER

GET SOURCE OPERAND FOR DOP'S

R-R EXECUTES (DOP WITH SM=Ø AND DM=Ø)

BRANCH (BRXX) EXECUTE

MOVE EXECUTES (SM=Ø)

FLOATING POINT ENTRY

REMAINING INSTRUCTIONS - FURTHER DECODE

GET DESTINATION OPERAND FOR DOP'S AND DOP'S WITH SM=Ø

R EXECUTES SOP WITH DM=Ø

JUMP EXECUTES DM=Ø-7

RESTORE FLOATING POINT CONSTANTS TO CSP

FURTHER DETAILS ON FOLLOWING PAGES
[TABLE A]

E

FIGURE 5-1
BASE MACHINE FLOW

FET∅1

```
BA ← PC
D ← PC+2
PC ← D
INITIATE:
DATICLKIR
```

FETCH
OVERLAP

FET∅2

```
MD ← DATA
IR ← DATA
```

FET∅3

```
BA ← PC
D ← PC+2
INITIATE:
DATI
```

ALLOW PREFETCH ? — YES

NO

INSTRUCTION PREFETCH ? — YES

NO

```
INHIBIT:
DATI
DEFEAT:
PC ← D
```

```
PC ← D
```

```
PC ← D
ALTER: DATI TO
DATICLKIR
```

```
MODE 6+7 FLOW
MD ← DATA
```

```
(ONE WORD)
EXECUTE
MD ← DATA
IR ← DATA
```

```
EXECUTE
```

```
EXECUTE
```

M.N.S 2/12/76

5-4

INSTRUCTION SEQUENCE:

MOV R∅, R1   #1, the Register-Register Intruction
Any A, B   #2, a non Register-Register Instruction

#1	FET∅1	FET∅2	FET∅3	MOV∅1	Clock CC ↓	
#2			FET∅1e	FET∅2e	FET∅3	execute

microcycle = 1   2   3   4   5   6

|— — — — — —|
      overlap
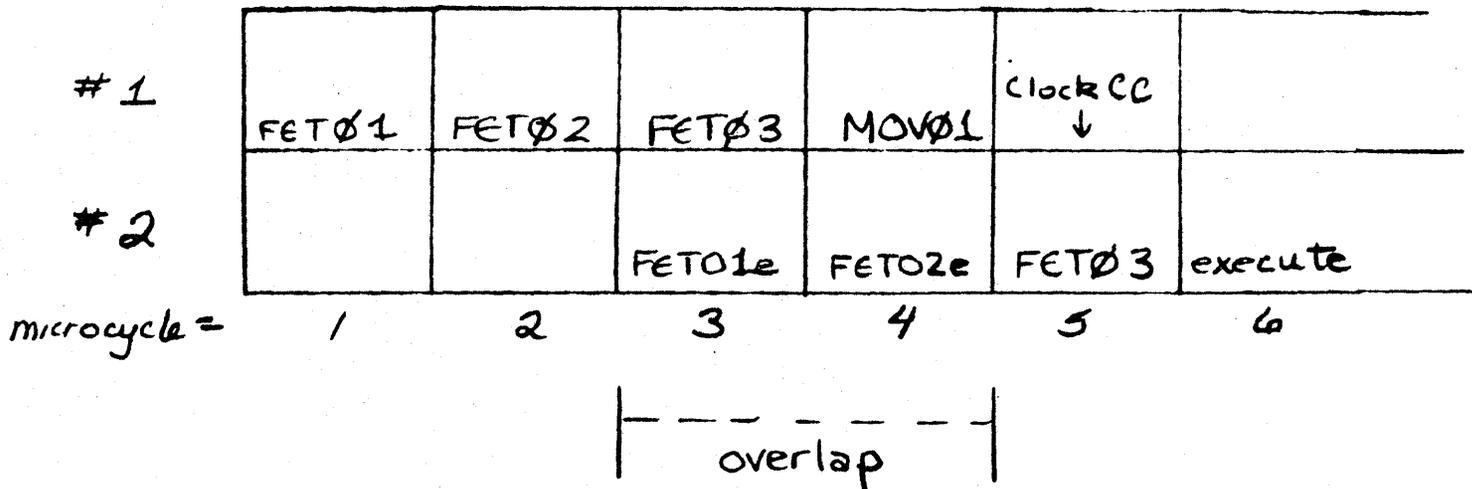
Overlap of Register-to-Register
with non Register-to-Register Instruction

INSTRUCTION SEQUENCE:

MOV RØ,R1   #1, first register - register instruction

MOV R1,R2   #2, second register - register instruction

MOV R2,R3   #3, third register - register instruction

	1	2	3	4	5	6	7	8	9
#1	FETØ1	FETØ2	FETØ3	R1←RØ MOVØ1	clock CC ↓				
#2			FETØ1e	FETØ2e	FETØ3	R2←R1 MOVØ1	clock CC ↓		
#3					FETØ1e	FETØ2e	FETØ3	R3←R2 MOVØ1	ClockCC ↓

microcycle =   1   2   3   4   5   6   7   8   9

|- - - - - - -|
↓
overlap #1-#2

|- - - - - - -|
↓
overlap #2-#3
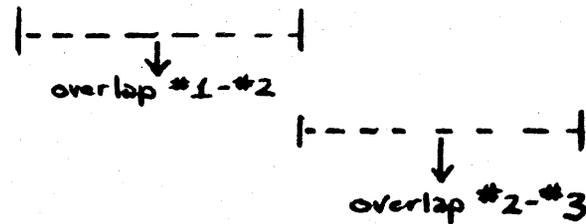
Overlap of Reg-Reg with another Reg-Reg Instruction

5-6

Figures 5-3 and 5-4 illustrate in more detail how the

overlapped fetch works. XFC instructions are never

overlapped, so the non-overlapped entry point to service

routines in the base machine should be used.

5.1.2   Instruction Decoding

Base machine decode is done in two steps: BUT(INSTR1) and

BUT(INSTR5). A large amount of logic is dedicated to this

initial IR decoding. Since this special purpose logic is not

tailored for XFC decoding, you will generally need to

do multi-step microprogrammed decoding. This method

is used by some sections of the base machine code such as the

status group of floating point instructions, which decode

IR<7:6>.

The instruction in the IR is also clocked into MD at P3 of

FET02. It can then be moved through the ALU, masked or

shifted in the shift tree, and then placed in the SR for a

CASE branch decode.

5.1.3   Instruction Execution

Execution of a PDP-11 instruction in the base machine
is usually done in one step. For example, the execution step
(E-phase) of an ADD instruction with source mode 0 and
destination mode 2 (ADD Rn,(Rm)+) requires the following:

```
P2,   D ← R(SF) PLUS MD     ! the destination calculation
      D(C) ← COUT15         ! put the correct data in MD
      DATO
      SET CONDITION CODES   ! clocking occurs in next
      J/BRA05               !uinstr, which will do a
                            ! BUT(SERVICE)
```

Because WCS routines are likely to be doing more complicated
activities in the datapath, multi-step execution will be
more common. The MUL instruction in EIS is an example of
multi=step execution: in addition to set-up steps, sixteen
shift-and-add steps are performed.

5.2   MICRO-LEVEL INTERRUPT ACTIVITIES

There are two methods by which the base machine handles
service and error conditions: Service and JAMUPP.

5.2.1   SERVICE

The service flow, which starts at SER01 or SER02, handles
non-fatal errors, interrupts, asynchronous errors, and WCS
micro-level service requests.
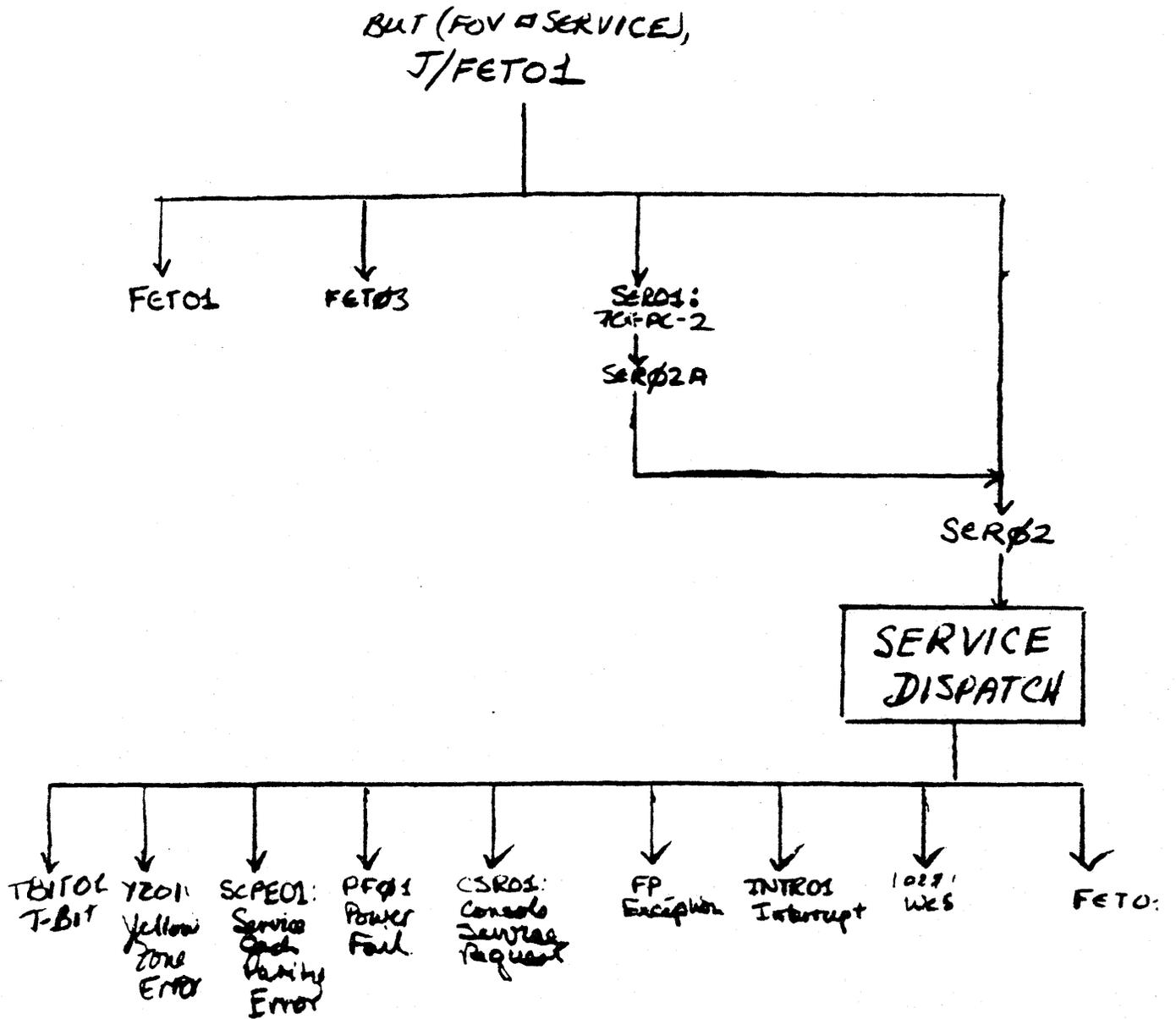
Figure 5-5   Service Dispatch

Each event requiring service sets a flag which is later

read by the service dispatch routine. If any of the flags

are set, any of the branches which test for service (e.g.

BUT(SERVICE)) will be true. A BUT on service is done at the

end of every macro-level instruction and at the end of the

shared trap flow.

Service conditions are handled in priority order. The

priority ranking is:

> Yellow Zone
> Cache Parity Error
> Power Fail
> Console Service Request
> Floating Point Exception
> Interrupt

Figure 5-5 shows the service flow.

5.2.2   JAMUPP

A JAM is a hardware-forced transfer of control to location 777,

which is the beginning of the JAM dispatch routine. In

general, those events which cause a JAM cannot be recovered

from, and therefore cause the macro instruction (including

XFC) currently being executed to be aborted.

However, a JAM is alos caused by a reference to internal

Unibus addresses, such as the KT or Cache registers. When

an internal address is specified at the micro-level with a

DATI or DATO, the following microinstruction should do nothing

except clock data into the CSP. The JAM routine will destroy

datapath state, and the timing of the hardware JAM is such

that the microword following the internal address reference

will, in effect, be executed twice. Thus no data manipulations

should be attempted in that microinstruction.

The JAMUPP routine services the following conditions:

>   Power-up
>   Internal Address
>   Microbreak
>   WCS Parity Error
>   Odd Address Error
>   Red Zone
>   KT Abort
>   Illegal Internal Address Reference
>   Cache Parity error
>   Unibus Timeout
>   Unibus Memory Parity Error

5.4   INTERFACE DEFINITIONS

5.4.1 Service

At the end of every macro-level instruction, or at least every

15 microseconds, a test for service must be performed. BUT(SERVICE),

issued when the UPF field contains the address of FET01,

causes the service routine to be invoked if needed. Service

starts (for non-overlapped fetch) starts at SER02 (0703).

The WCS user cannot use the same method the base machine

uses because the Page register must be clocked to jump to

page 0. Only BUT(SUBR B), BUT(SUBR A), and BUT(RETURN)

clock the page register.

To get around this, the user can finish execution with the

following branch:

```
        LAST:        BUT(SERVICE)
              ┌──────────┴──────────┐
              │    service          │    not service
              ▼                     ▼
        PAGE ◄─ 0             PAGE ◄─ 0
        BUT(SUBR B)          BUT(SUBR B)
        J/SER02              J/FET01
```

To eliminate this overhead, a location in the base machine

is provided to do the service test on a FET01 base. It is

called BRA05, at location 0003. Finish with:

```
        LAST:
              PAGE ◄─ 0
              BUT(SUBR B)
              J/BRA05
```

5.4.2   Generating a Trap

The 11/60 trap sequence begins at TRP00. It expects the trap vector to be in the MD when invoked.

For example:

```
        TRPA:
                BUT(CLEAR_FLAGS)        ! Select EMIT

          TRPB:
                EMIT/244,               !Generate trap vector
            P3, MD ← EMIT
                PAGE ← 0                !
                J/TRAP00                ! TRAP00 is at 0271
```

CHAPTER 6

WCS USAGE GUIDELINES

DRAFT

This chapter is intended to summarize the programming
conventions which will enable you to make effective use of the
Writable Control Store option, without damaging other
sections of the PDP-11/60.

6.1   WCS UNIBUS REGISTERS

You will use two Unibus locations to load the WCS array: the
WCS Address Register, WCSAR, and the WCS Data Register, WCSDR.

WCSAR has Unibus address 777542. Its format is shown in
Figure 6-1.
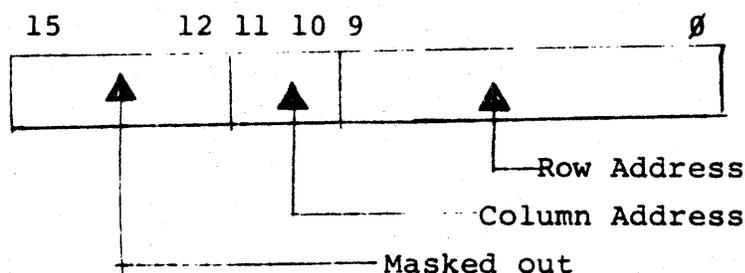
Figure 6-1   WGSAR Format

WCSDR has Unibus address 777544. A 16-bit word moved to this
address will be loaded into the WCS array at the location
specified by WCSAR. A 16-bit word read from WCSDR will come
from the array location specified by the current contents of
the WCS Address register.

Figure 6-2 shows a feasible user configuration of the WCS
address space. The three sections of page 7 are set aside
for local store use, while page 6 is used for control store.
The following program illustrates how one can use the Unibus
registers to load the microwords for that partitioning.
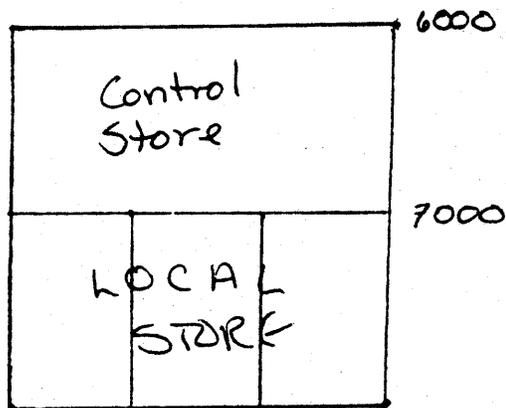
Figure 6-2 Possible User Configuration

This example assumes that the load image exists in main
memory as shown in Figure 6-3. The program loads the 1536
16-bit words (512 microwords) beginning at location LOADIM
in main memory into the control store, beginning at location
Ø (microaddress 6000_8).
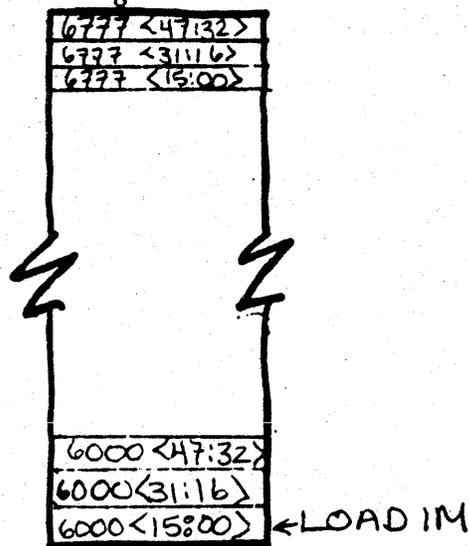
Figure 6-3 Load Image

```
TITLE WCSLD
FILENAME LOAD.MAC
CSECT
        .GLOBL LOADIM, WCSLD
        .IRPC X,012345
        R'X=%'X
        .ENDM
        SP=%6
        PC=%7
        WCSADR=177542
        WCSDR=177544

CSLD:   MOV     #LOADIM,R0          ;LOADIM IS STARTING ADDRESS OF ARRAY
        MOV     #512.,R1            ;LOAD PAGE 6 ONLY
        MOV     #000777,R2          ;MASK FOR INVERTING UPF
        CLR     @#WCSADR            ;START WITH ROW 0, COLUMN 0

OOP:    MOV     (R0)+,R3            ;MOVE LOW-ORDER WORD SO CAN XOR
        XOR     R2,R3               ;INVERTS UPF FIELD BITS
        MOV     R3,WCSDR            ;SEND TO ARRAY
        ADD     #2000,WCSADR        ;COLUMN 1 NOW
        MOV     (R0)+,WCSDR
        ADD     #2000,WCSADR        ;COLUMN 2 NOW
        MOV     (R0)+,WCSDR
        ADD     #4001,WCSADR        ;BACK TO COLUMN 0, ROW PLUS 1
        SOB     R1,LOOP
        .EXIT
        .END START
```

The WCS status register has Unibus Address 777540. Its format
is shown in Figure 6-4. It is provided for maintenance purposes.
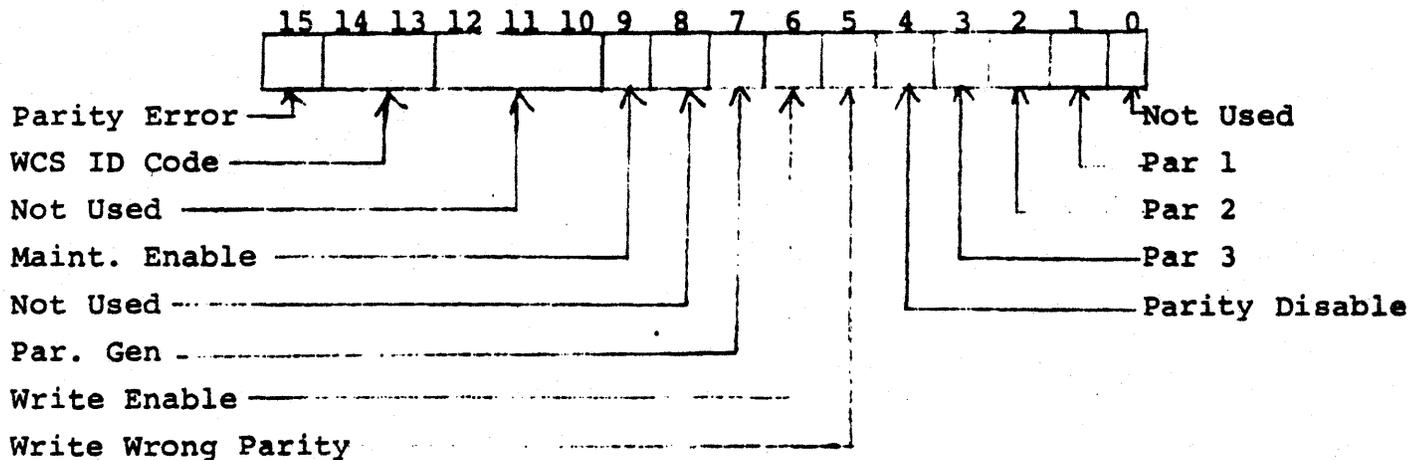
Figure 6-4   WCS Status Register

6.2   WCS Entry Points

There are many ways that control of the machine is passed into WCS. The following is a list of the entry points into the WCS address space and what the default instructions for each entry point are:

<u>ENTRY POINT</u>   <u>DESCRIPTION</u>

6000   WCS Microbreak Entry

                                     A microbreak occurs when the value loaded
                                     in the register is encountered and the
                                     microbreak enable bit is set. (FLAG<08>).
                                     Default response is to return to the console
                                     flow.

6001   XFC 076ØXX Dispatch

                                     This is a reserved instruction for DEC's
                                     future use. Default response is a reserved
                                     instruction trap. (trap vector 10)

6002   XFC 0767XX Dispatch
                                     User XFC Dispatch

                                     This is the entry point for the user's
                                     extended function codes. The user XFC
                                     (0767NX) is now further decoded according
                                     to bits 3-5 of the instruction to enter to
                                     one of the eight entries of the XFC dispatch
                                     table located at 6030.

6003   Reserved Instruction

                                     When the 11/60 executes one of the reserved
                                     instructions such as FIS or FASTx1 then
                                     control is passed here. Default response
                                     is a reserved instruction trap. (trap
                                     vector 10)

ENTRY POINT	DESCRIPTION

6004

ODD PC Dispatch

Whenever the base machine encounters a
New PC value of an interrupt or trap vector
which is odd then control is passed to this
point. Default response is to return into
the trap routine as if WCS was not present.
(TRP07)

6005

Default Service Condition Two

The Service Condition is checked once
between each macro instruction. If the
WCS Service bit of the flag register is
one (FLAG<07>) then control is passed to
this point. Default response is to
FET01.

6006

Default Jam Condition

When the XCS Extra Jam Pin is asserted low
and the internal suppressed clocks are
suppressed then control immediately passes to
this point. Default response is to go to the
the console flow.

6007

Default Service Condition One

Service passes control to this point if the
pin Extra Service is asserted. Default response
is to return to execute another instruction.

6010

Diagnostic Entry

When diagnose on the console is pressed
control passes to here. Default response
is to pass control to the End of Service
Routine.

ENTRY POINT	DESCRIPTION
6011 - 6015	XFC 0761XX through 0765XX Dispatch These instructions are reserved for DEC's future use. Default response is a reserved instruction trap. (trap vector 10)

Further explanation of the entry points can be found in the listing of resident section of WCS found in Appendix D.

6.3   TMS ROM ROUTINES FOR THE 11/60 WCS

This ROM resides on the WCS board and allows a program running
in the writeable control store of the 11/60 to use part of this
same control store as a block data store (local store). This
ability is realized by routines which perform block loads and
stores of various parts of the internal state of the 11/60. The
following portions of the machine are loaded or stored:

> General Registers
>
> Warm Floating Point Registers
>
> C Scratchpad except Base Constants
>
> User Scratch Registers
>
> Entire A Scratchpad
>
> Entire B Scratchpad
>
> Entire C Scratchpad

There are also routines to read and write one data item (with
and without loading the address register), read and write two
data items (with and without loading the address register),
read and write one data item indirectly.

Every TMS routine is invoked by a UCON function which loads the
Entry Point Register with the starting address of the TMS routine
wanted. All of the block move routines are entered with the WCS
local store address minus one (LSADR-1) clocked into D. Two null
cycles must be placed after the instruction that loads the Entry
Point Register. Return to the WCS routine will occur after the
function has been completed. The following example set of code
saves the general registers into local store address specified
in WCSB [0]-B:

```
EX1:
  P2-T,    D_B, WCSB[6]-B,        ! D ← local store address.
  NEXT,    J/EX2

EX2:
           TMSPTR_(STOREGRS),     ! Invoke GR store routine.
  NEXT,    J/EX3

EX3:
  NEXT,    J/EX4               ! First Null Word.

EX4:
  NEXT,    J/EX5               ! Second Null Word.
```

The following routines exist in the TMS ROM:

ROUTINE NAME	DESCRIPTION
READ	READ DATA
READANDINCR	READ DATA TO MD, INCREMENT ADDR
LOADANDREAD	LOAD ADDRESS AND THEN READ DATA
LOADREADINC	LOAD ADDRESS AND THEN READ DATA
WRITE	WRITE DATA
WRITEANDINC	WRITE DATA AND THEN INCREMENT ADDRESS
LOADANDWRITE	LOAD ADDRESS AND THEN WRITE DATA
LOADWRITEINC	LOAD ADDRESS, WRITE DATA, INCREMENT ADDRESS
INCANDREAD	INCREMENT ADDRESS AND THEN READ DATA
LOADADDRESS	LOAD ADDRESS
LOADGRS	LOAD FR'S FROM LOCAL STORE
STOREGRS	SAVE GR'S INTO LOCAL STORE
LOADFP	LOAD FP REGISTERS FROM LOCAL STORE
STOREFP	SAVE FP REGISTERS INTO LOCAL STORE
LOADCSP	LOAD CSP [00-13] INTO LOCAL STORE
STORECSP	SAVE CSP [00-13] INTO LOCAL STORE
LOADWCSAB	LOAD WCS WORK REGISTERS FROM LOCAL STORE
STOREWCSAB	SAVE WCS WORK REGISTERS INTO LOCAL STORE
SETLOAD	SAME AS LOADREADINC

6-8

ROUTINE NAME	DESCRIPTION
SETSTORE	SAME AS LOAD ADDRESS
ASPADLOAD	LOAD ASP[∅∅-37] FROM LOCAL STORE
ASPADSTORE	SAVE ASP[∅∅-37] INTO LOCAL STORE
BSPADLOAD	LOAD BSP[∅∅-37] FROM LOCAL STORE
BSPADSTORE	SAVE BSP[∅∅-37] INTO LOCAL STORE
ALLCSPLOAD	LOAD CSP[∅∅-17] FROM LOCAL STORE
ALLCSPSTORE	SAVE CSP[∅∅-17] INTO LOCAL STORE
LOADREADTWO	LOAD ADDRESS AND READ TWO PIECES OF DATA
INCREADTWO	INCREMENT ADDRESS AND READ TWO PIECES OF DATA
LOADWRITETWO	LOAD ADDRESS AND WRITE TWO PIECES OF DATA
WRITETWO	INCREMENT ADDRESS AND WRITE TWO PIECES OF DATA
READINDIRECT	READ DATA POINTED TO BY DATA
WRITEINDIRECT	WRITE DATA AT ADDRESS POINTED TO BY DATA (WCSA[∅])

More information on the TMS routines can be obtained from the
listing in Appendix E.

6.4 CAUTIONS AND WARNINGS

Because of the potential for problems due to microcode errors, certain microprogramming conventions must be followed. Many of these conventions have been mentioned in other sections of this manual; they are reiterated here for completeness.

6.4.1 Timing Considerations

6.4.1.1 Interrupt Latency -- In order to assure normal CPU responsiveness to interrupts, WCS microcode sequences should be implemented so that a maximum interval of fifteen (15) microseconds occurs between tests for interrupt service requests (BUT (SERVICE) or BUT (BG)). Note that a test for service occured just prior to the fetch of your XFC instruction.

One technique for making a long instruction interruptable is to "back-up" the processor state to the state at the beginning of the instruction when a service condition is detected. The processor state to be backed up includes; the Floating Point Accumulators;

R0 - R7; the KT-11 registers; the PSW.
Note that since the PC is pointing to the next instruction, it must be decremented. In this way, the aborted instruction may be attempted again after normal interrupt servicing is completed.

Some microinstructions can be restartable by keeping updatable paramemters in the general registers.

Notice that failing to check for service is effectively equivalent to setting the processor's priority level to 7.

6.4.2 UNIBUS Usage Conventions

Unibus control operations may not be performed withing microcode subroutines. BUS operations must not be performed in consecutive microwords.

6 — 10

6.4.3 Internal Scratchpad Use

The CSP Constants invalid flag must be set whenever the floating point constans are not in the CSP.

6.4.4 PDP-11 Processor State Requirements

The 11/60 microprocessor is used primarily to implement a PDP-11. Consequently, some constraints are not in the values of internal state permissible at the initiation of normal instruction fetch. The constraints include the following.

> The contents of ASP and BSP locations 0 through 7, the PDP-11 general registers, must have duplicate contents. ASP[0] = BSP[0], . . . ASP[1] = BSP[1], ASP[7] = BSP[7].

> The three locations in the CSP which contain the basic machine constants must, in fact, contain those constants.
> > CSP[17] = 1
> > CSP[16] = 0
> > CSP[14] = 2

6.4.5 Complete Decoding of Opcode Groups

It is expected that within each XFC opcode group, not all of the possible code combinations will have interpretations. Insturction decoding must be complete in the sense that those opcode values which do not have an interpretation result in an illegal instruction trap. Failure to provide this complete decoding could result in a loss of control due to a macro-level coding error.

CHAPTER 7

EXAMPLES

DRAFT

This chapter will provide examples of techniques and
applications for WCS microprograms.

7.1   BLOCK MOVE

In Chapter 1, the efficiency of a microcode implementation
of a BLOCK MOVE instruction was discussed. This section
examines the step-by-step implementation of such an
instruction. Since this example is intended to illustrate
a variety of concepts and procedures,it does not represent
the optimal implementation of such an instruction.

7.1.1   Instruction Specification

Define the BLOCK MOVE instruction as follows:

```
              15                    2 1 0
WORD 1:      | XFC.BLKMOV   | R_i |
WORD 2:      |          A          |
WORD 3:      |          B          |
```
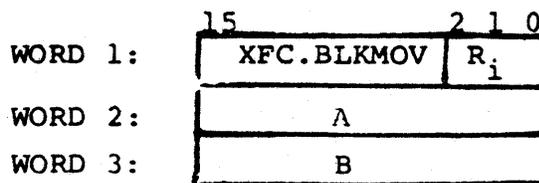
where:

XFC.BLKMOV	is the opcode for BLOCK MOVE
R_i	is the general register which contains the count in $R_i <07:00>$ ($\emptyset$ means 256)
A	is the starting source address
B	is the starting destination address

The format for this instruction is BLKMOV R_i, A, B.

For simplicity, this example avoids dealing with several problems,
e.g., a check for interrupts and setting the condition codes.

7.1.2   Specify Algorithm

The algorithm used in this implementation of BLOCK MOVE is:

        Set up Count
        Fetch A address
        Fetch B address
        Do while Count ≠ 0
                Move A word to B address
                Count = Count - 1
                Increment A and B
        End

7.1.3   Specify State

At entry to the BLKMOV microcode routine, PC-2 points to the
XFC instruction; the PC points to A, and PC + 2 points to
B.   A BUT(CLR-FLAGS) was performed in FETØ2, so the RES register
has been cleared.
By convention, the PC will point to the next instruction
after the operands are fetched.

The WCS registers will be used for temporary storage as follows:

        WCSA[0]   contains A addrs
        WCSA[1]   contains B address

7.1.4   First-pass Coding

The following instructions indicate what is happening in the datapath during the ececution of the BLKMOV instruction.

```
        ! BLOCK MOVE INSTRUCTION
        ! PC POINTS TO A ADDRESS


BEGIN:
        P2, SR←NOT R(DF)              ! UPON ENTRY RES IS CLEARED
                                      ! ONES'S COMPLEMENT OF COUNT

SETUP1:
        P2, D←SR+1                    ! TWO'S COMPLEMENT OF COUNT

SETUP2:
        P3, WCSB[0]←D                 ! PUT COUNT ON B-BUS SIDE
        NEXT, J/SETUP3
!
!
SETUP3:
        CNTR←WCSB [0]<07:00>          ! LOAD CNTR

SETUP4:
        P1, BA←PC,                    ! INITIATE FETCH OF A ADDRESS
            DATI,
        P2, D←PC PLUS 2,              ! POINT PC TO B ADDRESS
        P3, PC←D                      !

SETUP5:
        P3, MD←DATA                   ! MD←A ADDRESS

SETUP6:
        P2, D←MD,                     ! PUT A ADDRESS INTO
        P3, WCSA[0]←D                 ! A SCRATCHPAD

SETUP7:
        P1, BA←PC,                    ! INITATE FETCH OF B ADDRESS
            DATI,
        P2, D←PC PLUS 2,              ! POINT TO NEXT MACRO
        P3, PC←D                      ! INSTRUCTION

SETUP8:
        P3, MD←DATA                   ! MD←B ADDRESS

SETUP9:
      , P2, D←MD                      ! PUT B ADDRESS INTO
        P3, WCSB[1]←D                 ! A SCRATCHPAD
```

```
      !
      ! MAIN PROGRAM LOOP
      ! FETCH VALUE FROM ONE AREA AND PUT INTO OTHER AREA
      !

! THIS WORD MUST BE ON A XXXXXØ BOUNDARY
STRTLOOP:
      P1, BA←WCSA[0],             ! INITIATE FETCH OF VALUE
         DATI,
      P2, D←WCSA[0] PLUS 2,       ! POINT TO NEXT VALUE
      P3, WCSA[0]←D

LOOP2:
      P3, MD←DATA                 ! VALUE ARRIVES

LOOP3:
      P1, BA←WCSA[1],             ! SET BA WITH ADDRESS
      P2, D←WCSA[1] PLUS 2,       ! POINT TO NEXT VALUE IN
      P3, WCSA[1]←D               ! B AREA

LOOP4:
      P2, D←MD,                   ! INITIATE WRITE OF VALUE
         DATA

LOOP5:
      NEXT, BUT(COUNT),           ! VALUE WRITTEN INTO MEMORY
         J/STRTLOOP               ! LOOP UNTIL COUNT IS OVER
                                  ! TARGETS ARE
                                  ! STRTLOOP: COUNT NOT DONE
                                  ! FINISH: COUNT DONE

! THIS WORD MUST BE ON AN XXXXX1 BOUNDRY.
FINISH:
      NEXT, BUT(SUBRA), PAGE(0),  ! RETURN TO TEST FOR SERVICE WITH
         J/BRA05                  ! FET01 AS A TARGET
```

7.1.5   Try to Condense the Code

After sketching out the microcode, your next step will
be to try to exploit the parallelism in the IMP datapath
to reduce the number of cycles and/or words.

Note that in the code above, SETUP4 does the same thing
as SETUP7, and that SETUP5, SETUP8, and LOOP2 are all
identical. One method for reducing the number of
microinstructions (not cycles) is to try to make
subroutines out of repeated sections of code. It is
not a good programming practice to make a one- or two-word
subroutine, because the problems are usually greater than
the benefits. However, to illustrate the process, we will
ignore that point.

Look at what happens if MD ← DATA is made into a subroutine,
call it SUB1. SUB1 must contain a BUT(RETURN). The three
instructions which jump to it must use a BUT(SUBR B) to
load the Return register with the correct next address,
i.e.:

        SETUP3   must load Return with SETUP5
        SETUP6   must load Return with SETUP8
        STRTLOOP must load Return with LOOP3.

Now, can this be done ? No, because both SETUP6 and
STRTLOOP use the ALU and the scratchpads. The microword fields
for RETURN, RETURNPAGE, aNd PAGE overlap the ALU and
scratchpad control fields, so the BUT(SUBR B) cannot be
specified.

7-5

Another potential candidate for reduction is LOOP4:
can the DATO be done in LOOP3 where the BA is set up ?
Again, this reduction will not work because the D register is
needed for incrementing the B address as well as for the
Data Out.

The preceding unsuccessful search for words to eliminate
illustrates two of the three kinds of conflicts which
put constraints on your microprogramming. Microword
bit conflicts cause one level of constraint; datapath
components provide another. The third constraint,
timing, is somewhat easier to foresee.

7.1.6   Check For Interrupt Latency

The size limit on this block move is 255 words,
determined by the width of the CNTR. Each word that is
moved requires two memory cycles, a Data In and a Data Out.

For a worst case analysis, assume that there are no
cache hits, and that the memory cycle time is one
microsecond. Each iteration of the loop will then take approx-
imately two microseconds: 15 microseconds will have elapsed
when 7 words have been moved. Hence the 11/60's interrupt
latency rule will be violated if the block move instruction
cannot be interrupted.

One solution is to change the instruction specification so
that the state of the execution is held in the general
registers, namely: the decremented count, the incremented

A address, and the incremented B address. This would make the
instruction restartable as well as interruptable. The BLKMOV
microcode would test for Service once in each loop. When the
service condition arises the PC must be backed up to point to
the XFC instruction. The microprogrammer has two options when
the interrupt has been serviced:
1) return to XFC code and restart instruction from the beginning
2) set a flag internally and restart execution in the middle of
    the instruction.

This example also raises another point which the microprogrammer
must resolve and that is the problem of leaving an instruction in
a half executed fashion. (i.e., part of the B address field has
been changed) There are no PDP-11 instruction which leave things half
done. The microprogrammer may want to add a level of sophistication
and set a flag (Semaphore) at the macro level which signifies that
the B address field is invalid. (Does not contain old or new data.)

If the microprogrammer decides to restart execution of an instruction
in the middle he must either be sure that his task is the only one
using this instruction or else stack up the data he needs to know
insuring a valid implementation.

APPENDIX A
GLOSSARY

Adder
A device whose output is a representation of the sum of the
quantities represented by its inputs.

Address assignment
The allocation of an absolute address or a relative address
to a symbolic address.

ALU
Arithmetic and logic unit: a device whose output is a
representation of the result of the operation specified
by its control inputs performed upon the quantities
represented by its operand inputs.

Architecture
That set of a computer's features that are visible to the
programmer.

Barrel shifter

Bit Steering
An encoding technique in which one bit in the microword
is used to specify how bits in other fields are to be
interpreted.

Branch set
A set of microinstruction addresses which are potential
targets of a conditional branch.

Bus

A) A path over which information is transmitted from any of several sources to any of several destinations. B) A path over which information is transmitted from any of several sources to a single destination. A bus is a communications path which is capable of multiplexed use.

Chained sequencing

A method of instruction sequencing in which each instruction explicitly idetifies the next instruction to be executed; that is, it contains a separate address field. Contrast with instruction-counter sequencing.

Clock (verb)

To provide a signal to a register or other logic device which causes the data at its inputs to appear at its outputs. (Con trast latch.)

Combinational logic element

A device having at least one output channel and zero or more input channels, all characterized by discrete states, such that the state of each output channel is completely determined by the contemporaneous states of the input channels.

Constructed address

An instruction address that is formed by isolating the next-address field and modifying it with machine-state indicators by means of an arithmetic or logical operation.

Control field

A microword field which specifies an operation to be performed.

Control line

An input channel that controls the operation of a device
or logic element.

Control signal

A signal on a control line.

Control Storage

Memory in which executable microcode can be stored.

CPU

Central processing unit.

Cross-assembler

An assembler which executes on one machine and produces
machine language code for another machine.

Direct control

A method of organizing the micrword in which there is a one-
to-one mapping between bits in the microword and control
signals in the computer. Also known as unpacked control.

Disable

Emit field

A microinstruction field which provides either a data literal
to the datapath, or an address literal to the sequencing logic.

Emulation

The use of microprogramming techniques for the interpretive
execution of one machine by another.

Enable

Encoded control
A method of microword organization in which the values of
the control fields must be decoded to generate control signals.

Fetch
To obtain data or instructions from storage.

Firmware
A term used to describe the microrpogramming level, between
hardware and software, in the implementation of computer
systems. Also used to characterize program code which resides
in non-alterable, non-volatile memory, usually ROM.

Horizontal architecture
A loose term used to categorize machines whose microword has
some of the following attributes:
    A) it is capable of specifying multiple simultaneous
       operations;
    B) it is not highly encoded
    C) it is relatively wide; and
    D) it specifies the address of its successor

Host machine
A microprogrammable computer upon which an emulator for
a target machine is implemented.

Instruction-counter sequencing
A method of instruction sequencing in which a special counter
is used to store the address of the next instruction to be
executed. Most macroprograms are sequenced this way.

Instruction decode

The first phase of instruction interpretation, in which the
fields of the instruction are decoded to determine the
operations specified by the instruction.

Instruction register

A special-purpose register which stores only instructions;
generally serves as the source for instruction decode.

Interpretive execution

A method of implementation in which the execution of a·
single instruction at one level of the machine requires the
invocation and execution of multiple instructions for a
lower level of the machine.

Interrupt latency

The longest period of time a microprogram should execute
before allowing interrupts to be serviced.

Latch (verb)

To provide a signal to a register or other device which
causes the data at its outputs to take a constant value,
and to cease tacking changes in its input data.

Local storage

Data storage within a processor which is not accessed over
a main, general-purpose memory bus.

Macroinstruction

A machine language or macro-level instruction

Macro-level machine

The computer defined by the macro-level architecture.

Macromachine

Synonym for macro-level machine.

Masking

A programming technique; the first step in the process of extracting a non-word group from a word.

Microcode

A) One or more microinstructions, B) To write one or more microinstructions.

Microcycle

The smallest unit of time available for the execution of a single microinstruction.

Microinstruction

An instruction which causes the generation of control signals to control the logical elements of a processor.

Micro-operation

An operation specified by a control field of a microinstruction.

Microprocessor

A) A processor on an LSI chip, usually implemented in MOS, Bipolar, or I^2L technology. B) That portion of a central processing unit that interprets and executes microcode.

Microprogram

A microcode routine; a program composed of microinstructions.

Microprogrammable
Pertaining to the capability to control the actions of the
micro-level machine via microprogramming.

Microword
A word of control storage.

Organization
A level below architecture, organization is concerned with
how the facilities available to the programmer are provided.

PROM
Programmable read-only memory.

PROM blaster
The device used to program a PROM.

ROM
Read-only memory. A storage device whose contents cannot
be altered.

Realization
In the hierarchy of architectures and organizations, realization
describes the lowest level -- the chips and wires which
implement a machine organization.

Reentrant program
A program that can be interrupted at any point, and then
resumed from the point where it was interrupted.

Register

Scratchpad memory
Local storage within the central datapaths of a machine.

Scratchpad registers
Individual words of a scratchpad memory.

Shift register
A register capable of shifting its contents to the right
or left when a control signal is received.

Special-purpose register
A register whose use is limited to a special purpose, such
as a floating-point register, an instruction register, or
a stack pointer.

Set-up register
A register used to store relatively static control information.
After loading, the data in the register can be used to
supplement the control information provided by the micro-
instruction.

Target machine
The computer whose architecture is implemented by an
emulator running on a host machine.

Tri-state logic
A type of logic in which the source that controls a given line
can force the line into one of three states:
    A) Logical one
    B) Logical zero
    C) Off, or high impedance state.
In the off, or high impedance state, the line is available for
other devices to put information on it without affecting the
original source that drives the line. Hence a selecting (or
multiplexing) function can be realized.

Vertical architecture

A general term used to describe machines which have some of the following attributes:

    A) Instruction-counter sequencing

    B) Relatively narrow microwords

    C) Microinstructions specify a single operation

    D) The microinstruction is highly encoded.

APPENDIX B

INSTRUCTION SET PROCESSOR (ISP) NOTATION

The Instruction Set Processor (ISP) notation provides a
coherent method of describing hardware operations. A subset o
of the ISP notation has been used in examples throughout
this manual. This appendix described the elements of this
subset, and the conventions which are followed in its use.

ᵀor a complete description of ISP notation, refer to paegs
15-36 and the Appendix of Computer Structures: Readings
and Examples[1].

B.1   MEMORY DECLARATIONS

In this manual, memory declarations have the general form:

$$M[a:b]\langle x:y \rangle$$

where:

    M is the name of the declared entity

    a and b   are the upper and lower bounds (addresses)
           of the memory

    :   indicates a range of values

    x and y   are the upper and lower bounds of the
           elements (bits) of the memory.

B.1.1.   Conventions

The ISP notation provides for mixed numbering systems by
means of subscripts, e.g.: $M[a:b_8]\langle x:y \rangle_{10}$.
On the 11/60, these explicit subscripts are omitted, and
the following set of conventions is followed.

---

[1]   Bell, C.G., and Newell, Allen: Computer Structures: Readings
and Examples . McGraw-Hill; New York, 1971.

Locations are numbered in octal, and are li.sted in ascending order. Bits are numbered in decimal , listed from leftmost to rightmost.

Thus,   ASP[0:17]<15:0> declares ASPLO to be a 16-word memory. Each word is composed of 16 bits named 0, 1, 2,...15.

B.2   ASSIGNMENT AND SUBSTITUTION

The "colon_equals" symbol (:=) assigns a name to an expression. Thus, x:=y assigns the name x to mean the same thing as expression y.

A slash mark is used to indicate abbreviation or replacement. Thus, if x is any name, and y is any name, the x/y assigns y as a synonym for x.

To illustrate the use of these symbols, let's look at one of the fields in the microword, BEN.

```
        BEN :=μ<43:42>
        BASCON  := 0
        CSP  ;= 1
        BSPLO := 2
        BSPHI := 3
```

With these definitions and assignments, we can specify the BEN field value for a particular microinstruction with:
                BEN/BASCON
This specification indicates that the field composed of bits 43 and 42 of the microword is to be loaded with the value zero.

B.3   OPERATIONS

Most of the symbols for operations in ISP are used widely
enough that they are self-explanatory. There are two
symbols, however, which may be new to some programmers.

A back arrow $(\leftarrow)$ indicates the reading, transmission,
and writing of data. For example,

$$t_1 \leftarrow t_2$$

indicates that t_1 receives t_2. If t_2 is a memory, then
T_1 receives t_2's contents. If t_2 is a value, then the value
is put in t_1.

    CSP[2] $\leftarrow$ 200   ! The value 200 is placed in CSP[2]

    ASPLO[0] $\leftarrow$ D   ! The contents of D is placed
                                     in ASP[0].

A front arrow, $(\rightarrow)$, indicates a control operation which
invokes an action-sequence. Thus,

    b $\rightarrow$ action-sequnce

indicates that if b is true, then the action-sequence is
applied; otherwise, it is ignored.   For example:

$$SR\langle 4:0\rangle \neq 0 \rightarrow Branch$$

B.4   ISP NOTATION SUMMARY

DRAFT

APPENDIX C

BIBLIOGRAPHY

Bell, C.G., and Newell, Allen, Computer Structures: Readings and Examples. McGraw-Hill, New York, 1971.

Gear, C.W., Computer Organization and Programming, 2nd Edition, New York: McGraw-Hill, 1974. Chapter 7.

Husson, S.S., Microprogramming: Principles and Practices, Englewood Cliffs: Prentice-Hall, 1970.

Flynn, M.J., "Microprogramming - Another Look at Internal Computer Control," Proceedings of the IEEE, Vol. 63, Nov. 1975, pp. 1554-1567.

Flynn, M.J., and ROSIN, R.F., "Microprogramming:An Introduction and a Viewpoint," IEEE Trans. Comput., Vol C-20, July 1971. pp. 727-731.

Preprints of the Seventh Annual Workshop on Microprogramming

Rosin, R.F., "Contemporary Concepts of Microprogramming and Emulation," Computing survey, 1, 4, Dec. 1969.

Salisbury, Allan B, Microprogrammable Computer Architectures, New York: American Elsevier Publishing Co., 1976.

APPENDIX D

BASIC SKELETON OF RESIDENT SECTION FOR WCS OPTION OF THE PDP 11/60

THERE ARE NINE CLASSES OF ENTRY POINTS INTO THE BASIC RESIDENT
SECTION.   THESE ENTRY POINTS AND THERE REASONS ARE LISTED
HERE:

(1)   ERROR ROUTINE

        6016   USER HAS LOST CONTROL OF MICROCODE OR OBTAINED
                ILLEGAL ERROR CONDITION.

(2)   NON-USER XFC DISPATCH

        6001   XFC 0760NN
        6011   XFC 0761NN
        6012   XFC 0762NN
        6013   XFC 0763NN
        6014   XFC 0764NN
        6015   XFC 0765NN

(3)   USER XFC DISPATCH

        6002   XFC 0767NN

(4)   ODD PC DISPATCH

        6004   TRAP VECTOR OR INTERRUPT VECTOR CONTAINS AN ODD ADDRESS.

(5)   WCS MICROBREAK

        6000   MICROBREAK HAS BEEN ENABLED AND ADDRESS HAS BEEN MATCHED

(6)   RESERVED INSTRUCTIONS

        6003   A RESERVED INSTRUCTION HAS BEEN EXECUTED.
        (FIS,FASTX1,ETC.)

(7)   DEFAULT SERVICE CONDITIONS

        6005   WCS SERVICE (FLAG) HAS BEEN TURNED ON BY MICROCODE.
        6007   PIN EXTRA SERVICE HAS BEEN PULLED DOWN.

(8)   DEFAULT JAM CONDITION

        6006   EXTERNAL JAM PIN HAS BEEN PULLED.

(9)   DIAGNOSTIC CONDITION

        6007   DIAGNOSE CONDITION PRESSED ON THE CONSOLE.

CODE

ISFCH: ITARGETS ARE ENTRIES IN DISPATCH TABLE.
BEGIN=000[6030:60371

```
143
144   I
145   I          THE FOLLOWING CODE IMPLEMENTS THE BASIC RESIDENT FUNCTIONS
146   I          NEEDED TO SERVE THE WCS USER.   NOTE THAT THE USER IF HE CHOOSE
147   I          CAN OVERLAY THE PORTIONS OF THIS CODE WHICH HE NEEDS TOO.
148   I
149   I
150
151
152   .TOC     RESIDENT ROUTINE CODE.
153
154
155   I
156   I          WCS MICROBREAK ENTRY POSITION
157   I
158   I          A MICROBREAK OCCURS WHEN THE VALUE LOADED IN THE MICROBREAK
159   I          REGISTER IS ENCOUNTERED AND THF MICROBREAK ENABLE BIT IS SET,
160   I          (FLAG <08>).  IF THE TRAP ON MICROBREAK BIT (WHAMI <09>) IS ON
161   I          THEN THE BASE MACHINE TRAPS TO TRAP VECTOR 4.   CONTROL ONLY
162   I          COMES TO THIS LOCATION IF THAT BIT IS OFF.
163   I
164   I          THE FOLLOWING INPUT CONDITIONS ARE OF INTEREST:
165   I
166   I          (1) MICROBREAK ENABLE BIT (FLAG<08>) HAS BEEN CLEARED.
167   I          (2) LOG ROUTINE HAS LOGGED.
168   I              (A) CSP INVALID BIT SET IN FLAG REGISTER.
169   I              (B) CSP(0) <-- JAM REGISTER
170   I                  CSP(1) <-- STATUS XOR 340
171   I                  CSP(2) <-- PBA
172   I                  CSP(3) <-- CUA
173   I                  CSP(4) <-- FLAGS # VECTOR
174   I                  CSP(5) <-- WHAMI
175   I                  CSP(6) <-- CACHE DATA
176   I                  CSP(7) <-- HITTAG WITH TAG FIELD HIGH.
177   I                  CSP(11) <-- OS REGISTER.
178   I
179   I          IN THE DEFAULT CASE CONTROL RETURNS TO THE CONSOLE.
180   I
181
182
183   XFCSUB01:
184      NEXT, PAGE(1),BUT(SUBRB),              IRETURN TO FETCH NEXT INSTRUCT\
185           J/CON99
      6000   0 00000000 00000000 01000000 00000000 00111000 00100000
186
187
188   I
189   I          XFC      0760XX DISPATCH
190   I
191   I          THIS XFC INSTRUCTION IS RESERVED BY DEC AND NOT TO
192   I          BE USED BY THE WCS PROGRAMMER.   DEFAULT IS JUMP TO THE TRAP ROUTI\
193   I          TO INITIATE A RESERVED INSTRUCTION TRAP. (TRAP VECTOR 10).
194   I
195   I
196
197
198   XFCOTH01:
199      P3,   WR-CSP,CSPD(D15),EMIT/0010,       I10 - FOR TRAP VECTOR.
200      NEXT, PAGE(0),BUT(SUBRB),              IRETURN TO TRAP FLOW.  PAGE 0I
201           J/TRP00                           ISHARED WITH EMIT BITS.
      6001   0 00001000 00000010 00000000 00101000 00111000 01010111
```

D-2

```
12
13
14    |
15    |          XFC       0767NX DISPATCH
16    |          USER XFC DISPATCH
17    |
18    |          THIS IS THE ENTRY POINT FOR ALL OF THE USERS INSTRUCTIONS.
19    |          THE USER XFC (0767NX) IS NOW FURTHER DECODED ACCORDING
10    |          TO BITS 3-5 OF THE INSTRUCTION TO ENTER ONE OF THE EIGHT
11    |          ENTRIES OF THE XFC DISPATCH TABLE.
12    |
13    |          THE FOLLOWING INPUT CONDITIONS EXIST AT THIS TIME:
14    |
15    |          (1)     MD CONTAINS XFC INSTRUCTION.
16    |          (2)     SR CONTAINS XFC INSTRUCTION.
17    |          (3)     TR CONTAINS XFC INSTRUCTION.
18    |
19    |
10
11
12  USERDISP01:
13     NEXT, BUT(IR5-3),                           |JUMP TO ONE OF THE EIGHT
14          J/USER0                                |ENTRIES IN THE DISPATCH TABLE.
      6002  0  00000000 00000000 00000000 00000000 00001010 00011000
15
16
17    |
18    |          RESERVED INSTRUCTIONS
19    |
10    |          THE PDP 11/60 CONTAINS A FEW RESERVED INSTRUCTIONS WHICH
11    |          ARE NOT IMPLEMENTED SUCH AS FIS AND FASTX.   IF THE WCS
12    |          IS ENABLED THEN EXIT IS TAKEN TO THIS SPOT.   THE DEFAULT
13    |          RESPONSE IS TO EXECUTE A RESERVED INSTRUCTION TRAP. (ERROR
14    |          VECTOR 10).
15    |
16    |          THE FOLLOWING INPUT CONDITIONS EXIST AT THIS TIME:
17    |
18    |          (1)     D REGISTER CONTAINS WHAMI VALUE.
19    |          (2)     D(C) CONTAINS ALU<07>.
10    |
11    |
12
13  WCSRSVD01:
14     P3,    CSPD(MD),EMIT(0010),                 |10 - FOR TRAP VECTOR.
15     NEXT, PAGE(0),BUT(SUBRB),                    |RETURN TO TRAP FLOW TO
16          J/TRP30                                 |INITIATE TRAP.
      6003  0  00001000 00000010 00000000 00101000 00111000 01010111
17
18
19
50    |
51    |          ODD PC DISPATCH
52    |
53    |          WHENEVER THE BASE MACHINE FINDS AN ODD ADDRESS IN AN INTERRUPT
54    |          OR TRAP VECTOR AND THE WCS IS PRESENT IT EXITS TO
55    |          THIS LOCATION.   THIS ALLOWS THE USER TO WRITE A FAST INTERRUPT
56    |          HANDLER CONSISTING OF WCS MICROCODE.   THE DEFAULT CASE WILL
57    |          BE TO RETURN TO THE TRAP FLOW JUST AS IF WCS WAS NOT PRESENT.
```

THE FOLLOWING INPUT CONDITIONS EXIST UPON ENTRY TO THIS
        LOCATION:

        (1) R(VECT) CONTAINS NEW PC VALUE IN TRAP VECTOR (ODD VALUE).
        (2) D CONTAINS NEW PS VALUE.
        (3) SR CONTAINS OLD PS VALUE.

        THERE IS NO WAY TO FIND OUT WHICH TRAP VECTOR WAS ENCOUNTERED.
        USER MICROCODE WILL HAVE TO ANALYZE THE NEW PC OR PS VALUE
        TO DETERMINE THE DEVICE WHICH INTERRUPTED.

ODDPCDIS01:
    NEXT, PAGE(4),BUT(SUBRB),   ;RETURN RIGHT BACK TO THE
         J/TRP07   ;BASE MACHINE TRAP FLOW.
6004   0 00000000 00000001 00000000 00000000 00111001 10011001

        DEFAULT SERVICE CONDITION TWO

        THE SERVICE CONDITION IS CHECKED ONCE BETWEEN EACH MACRO
        INSTRUCTION. ONE CONDITION THAT CAUSES SERVICE TO BE
        NEEDED IS IF THE WCS SERVICE BIT OF THE FLAG REGISTER IS
        TURNED ON. (FLAG <07>).   THE WCS SERVICE BIT IS TURNED
        ON BY THE WCS MICROPROGRAMMER WHENEVER HE WISHES TO MONITOR
        (TRACE) THE NEXT INSTRUCTIONS.   WHEN THIS BIT IS ON CONTROL
        IS PASSED TO THIS INSTRUCTION ONCE BETWEEN EVERY MACRO
        INSTRUCTION.   IF THIS BIT IS ON THEN THE WCS CAN MONITOR
        EVERY INSTRUCTION EXECUTED.   THE DEFAULT IS TO NOT
        MONITOR ANYTHING BUT INSTEAD RETURN TO EXECUTE ANOTHER
        INSTRUCTION. (FET01) NOTE THAT RETURN MUST NOT BE
        TO BRA05 AS AN INFINITE LOOP WILL RESULT.

SVCDEFLT02:
    NEXT, PAGE(0),BUT(SUBRA),   ;RETURN TO EXECUTE
         J/FET01   ;THE NEXT INSTRUCTION.
6005   0 00000000 00000000 00000000 00000000 00111011 11000010

        DEFAULT JAM CONDITION

        WHEN THE XCS EXTRA JAM PIN IS YANKED THIS CAUSES
        AN IMMEDIATE JAM CONDITION TO OCCUR. IF THE WCS IS ENABLED
        THEN CONTROL IS PASSED TO THIS LOCATION.   IN THE DEFAULT CASE
        CONTROL IS RETURNED TO THE CONSOLE FLOW.

JAMDEFLT01:
    NEXT, PAGE(1),BUT(SUBRB),   ;RETURN TO CONSOLE CONTROL.
         J/CON99
6006   0 00000000 00000000 01000000 00000000 00111000 00100000

```
|
|           DEFAULT SERVICE CONDITION ONE
|
|           WHEN THE PIN EXTRA SERVICE IS PULLED THEN THIS
|           CAUSES A SERVICE CONDITION TO ARISE.   EXIT IS TO THIS
|           ROUTINE TO HANDLE THAT CONDITION WHEN WCS IS ENABLED.
|           THE DEFAULT CASE IS TO RETURN TO EXECUTE ANOTHER
|           INSTRUCTION.
|
|
SVCDEFLT01:
   NEXT, PAGE(0),BUT(SUBRB),                  |RETURN TO EXECUTE
         J/FET01                              |THE NEXT INSTRUCTION.
   6007   0 00000000 00000000 00000000 00000000 00111001 11000010

|
|           DIAGNOSTIC ENTRY
|
|           WHEN DIAGNOSE ON THE CONSOLE IS PRESSED AND WCS IS
|           ENABLED THEN CONTROL IS PASSED TO THIS POINT.   DEFAULT
|           RESPONSE IS TO PASS CONTROL TO THE END OF SERVICE ROUTINE.
|
|
WCSDIAG01:
   NEXT, PAGE(1),BUT(SUBRB),                  |RETURN TO END
         J/EOS1A                              |OF SERVICE ROUTINE.
   6010   0 00000000 00000000 01000000 00000000 00111001 00110000


|
|           XFC     0761XX DISPATCH
|
|           THIS XFC INSTRUCTION IS RESERVED BY DEC AND NOT TO BE USED
|           BY THE WCS PROGRAMMER.   DEFAULT IS TO JUMP TO THE TRAP
|           ROUTINE TO INITIATE A RESERVED INSTRUCTION TRAP. (TRAP VECTOR 10).
|
|


XFCOTH10:
   P3,   WR-CSP,CSPD(D15),EMIT/0010,          |10 - FOR TRAP VECTOR.
   NEXT, PAGE(0),BUT(SUBRB),                  |RETURN TO TRAP FLOW.  PAGE BITS
         J/TRP00                              |SHARED WITH EMIT BITS.
   6011   0 00001000 00000010 00000000 00101000 00111000 01010111

|
|           XFC     0762XX DISPATCH
|
|           THIS XFC INSTRUCTION IS RESERVED BY DEC AND NOT TO BE USED
|           BY THE WCS PROGRAMMER.   DEFAULT IS TO JUMP TO THE TRAP
|           ROUTINE TO INITIATE A RESERVED INSTRUCTION TRAP. (TRAP VECTOR 10).
|
|



XFCOTH20:
   P3,   WR-CSP,CSPD(D15),EMIT/0010,          |10 - FOR TRAP VECTOR.
   NEXT, PAGE(0),BUT(SUBRB),                  |RETURN TO TRAP FLOW. PAGE BITS
         J/TRP00                              |SHARED WITH EMIT BITS.
   6012   0 00001000 00000010 00000000 00101000 00111000 01010111
```

(D-5)

```
|
|          XFC      0763XX  DISPATCH                              ,
|
|       THIS XFC INSTRUCTION IS RESERVED BY DEC AND NOT TO BE USED
|       BY THE WCS MICROPROGRAMMER.   DEFAULT IS TO JUMP TO THE TRAP
|       ROUTINE TO INITIATE A RESERVED INSTRUCTION TRAP. (TRAP VECTOR  10).
|
|


XFCOTH30:
   P3,    WR-CSP,CSPD(D15),EMIT/0010,      |10 - FOR TRAP VECTOR.
   NEXT, PAGE(0),BUT(SUBRB),               |RETURN TO TRAP FLOW.  PAGE BITS
         J/TRP00                           |SHARED WITH EMIT BITS.
6013   0 00001000 00000010 00000000 00101000 00111000 01010111


|
|          XFC      0764XX  DISPATCH
|
|       THIS XFC INSTRUCTION IS RESERVED BY DEC AND NOT TO BE USED
|       BY THE WCS MICROPROGRAMMER.   DEFAULT IS TO JUMP TO THE TRAP
|       ROUTINE TO INITIATE A RESERVED INSTRUCTION TRAP. (TRAP VECTOR 10).
|
|


XFCOTH40:
   P3,    WR-CSP,CSPD(D15),EMIT/0010,      |10 - FOR TRAP VECTOR.
   NEXT, PAGE(0),BUT(SUBRB),               |RETURN TO TRAP FLOW.  PAGE BITS
         J/TRP00                           |SHARED WITH EMIT BITS.
6014   0 00001000 00000010 00000000 00101000 00111000 01010111


|
|          XFC      0765XX  DISPATCH
|
|       THIS XFC INSTRUCTION IS RESERVED BY DEC AND NOT TO BE USED
|       BY THE WCS MICROPROGRAMMER.   DEFAULT IS TO JUMP TO THE TRAP
|       ROUTINE TO INITIATE A RESERVED INSTRUCTION TRAP. (TRAP VECTOR 10).
|
|


XFCOTH50:
   P3,    WR-CSP,CSPD(D15),EMIT/0010,      |10 - FOR TRAP VECTOR.
   NEXT, PAGE(0),BUT(SUBRB),               |RETURN TO TRAP FLOW.  PAGE BITS
         J/TRP00                           |SHARED WITH EMIT BITS.
6015   0 00001000 00000010 00000000 00101000 00111000 01010111


|


.
```

D-6

```
|        WCS ERROR ROUTINE
|
|        WHEN THE WCS RESIDENT IS LOADED ALL OTHER LOCATIONS OF
|        WCS WILL CONTAIN A JUMP TO THIS LOCATION.   THE USERS
|        MICROCODE WILL BE OVERLAID ON TOP OF THESE JUMPS.   WHENEVER
| -      THE USERS MICROCODE LOSES CONTROL (ENTERS UNPLANNED ADDRESS)
|        OR DECIDES A FATAL ERROR HAS OCCURRED CONTROL WILL ENTER
|        HERE.   DEFAULT HANDLING FOR THIS CASE IS TO EXIT TO THE
|        CONSOLE CODE JUST AS IF A HALT INSTRUCTION WAS ENCOUNTERED.
|        IN A LOT OF CASES THE USER WILL WANT TO BUILD HIS
|        OWN ERROR ROUTINE.
|
|


WCSERR:
   NEXT, PAGE(1),BUT(SUBRB),              |RETURN TO BASE MACHINE.
        J/CON99                           |HALT FLOW.
,016   0 00000000 00000000 01000000 00000000 00111000 00100000
|



|
|        USER XFC DISPATCH TABLE (0767NX)
|
|        ACCORDING TO BITS 3-5 OF THE USER XFC INSTRUCTION ONE
|        OF THESE EIGHT ENTRIES OF THE DISPATCH TABLE IS ENTERED.
|        THIS DISPATCH TABLE IS MODIFIED BY THE MICROCODE LOADER
|        WHEN IT LOADS THE USERS MICROCODE ACCORDING TO THE ENTRY
|        STATEMENTS SPECIFIED IN THE ASSEMBLY.   THE DEFAULT FOR
|        EACH OF THESE ENTRIES IS TO RETURN TO THE BASE MACHINE AS
|        IF A RESERVED INSTRUCTION WAS ENCOUNTERED.   AN INSTRUCTION
|        IS ONLY ACCEPTED AS LEGAL IF WCS CODE TO EXECUTE THAT
|        INSTRUCTION HAS BEEN LOADED.   NOTE THAT THE USER MICROPROGRAMMER
|        CAN FURTHER DECODE BITS 0-2 OF THE XFC INSTRUCTION TO
|        FACILITATE MORE XFC INSTRUCTIONS.  SINCE THE XFC IS IN THE SR
|        FURTHER DECODE OF IT CAN OCCUR BY TESTING SR(2-0).
|
|


.CASE 1 OF DISPCH                         |XFC 07670X
USER0:
   P3,   WR-CSP,CSPD(D15),EMIT/0010,      |10 - RESERVED TRAP.
   NEXT, PAGE(0),BUT(SUBRB),              |RETURN TO TRAP FLOW.  PAGE BITS
        J/TRP00                           |SHARED WITH EMIT BITS.
,030   0 00001000 00000010 00000000 00101000 00111000 01010111

.CASE 2 OF DISPCH                         |XFC 07671X
USER1:
   P3,   WR-CSP,CSPD(D15),EMIT/0010,      |10 - RESERVED TRAP.
   NEXT, PAGE(0),BUT(SUBRB),              |RETURN TO TRAP FLOW.  PAGE BITS
        J/TRP00                           |SHARED WITH EMIT BITS.
,031   0 00001000 00000010 00000000 00101000 00111000 01010111

.CASE 3 OF DISPCH                         |XFC 0762X
USER2:
   *P3,  WR-CSP,CSPD(D15),EMIT/0010,      |10 - RESERVED TRAP VECTOR.
   NEXT, PAGE(0),BUT(SUBRB),              |RETURN TO TRAP FLOW.  PAGE BITS
        J/TRP00                           |SHARED WITH EMIT BITS.
6032   0 00001000 00000010 00000000 00101000 00111000 01010111
```

D-7

```
.CASE 4 OF DISPCH                         IXFC 07673X
USER3:
   P3,   WR-CSP,CSPD(D15),EMIT/0010,       110 - FOR TRAP VECTOR.
   NEXT, PAGE(0),BUT(SUBRB),              IRETURN TO TRAP FLOW. PAGE BITS
         J/TRP00
6033  0 00001000 00000010 00000000 00101000 00111000 01010111

.CASE 5 OF DISPCH                         IXFC 07675X
USER4:
   P3,   WR-CSP,CSPD(D15),EMIT/0010,       110 - FOR TRAP VECTOR.
   NEXT, PAGE(0),BUT(SUBRB),              IRETURN TO TRAP FLOW.  PAGE BITS
         J/TRP00                          ISHARED WITH EMIT BITS.
6034  0 00001000 00000010 00000000 00101000 00111000 01010111

.CASE 6 OF DISPCH                         IXFC 07675X
USER5:
   P3,   WR-CSP,CSPD(D15),EMIT/0010,       110 - FOR TRAP VECTOR.
   NEXT, PAGE(0),BUT(SUBRB),              IRETURN TO TRAP FLOW.  PAGE BITS
         J/TRP00                          ISHARED WITH EMIT BITS.
6035  0 00001000 00000010 00000000 00101000 00111000 01010111

.CASE 7 OF DISPCH
USER6:
   P3,   WR-CSP,CSPD(D15),EMIT/0010,       IXFC 07676X
   NEXT, PAGE(0),BUT(SUBRB),              IRETURN TO TRAP FLOW.  PAGE BITS
         J/TRP00                          ISHARED WITH EMIT BITS.
6036  0 00001000 00000010 00000000 00101000 00111000 01010111

.CASE 8 OF DISPCH                         IXFC 07677X
USER7:
   P3,   WR-CSP,CSPD(D15),EMIT/0010,       110 - FOR TRAP VECTOR.
   NEXT, PAGE(0),BUT(SUBRB),              IRETURN TO TRAP FLOW.  PAGE BITS
         J/TRP00                          ISHARED WITH EMIT BITS.
6037  0 00001000 00000010 00000000 00101000 00111000 01010111
```

D-8

TMS ROM MICROCODE FOR 11/60

THIS MICROCODE GOES INTO THE TMS ROM (TRANSFER MICROSTORE
ROM).   THIS ROM RESIDES ON THE WCS BOARD AND ALLOWS A PROGRAM
RUNNING IN THE WRITEABLE CONTROL STORE OF THE 11/60
TO USE PART OF THIS SAME CONTROL STORE AS A BLOCK DATA STORE.
(LOCAL STORE)   THIS ABILITY IS REALIZED BY ROUTINES WHICH
PERFORM BLOCK LOADS AND STORES OF VARIOUS PARTS OF THE INTERNAL
STATE OF THE 11/60.   THE FOLLOWING PORTIONS OF THE MACHINE
ARE LOADED OR STORED:

        (1) GENERAL REGISTERS
        (2) WARM FLOATING POINT REGISTERS
        (3) C SCRATCHPAD EXCEPT BASE CONSTANTS
        (4) USER SCRATCH REGISTERS
        (5) ENTIRE A SCRATCHPAD
        (6) ENTIRE B SCRATCHPAD
        (7) ENTIRE C SCRATCHPAD

THIS MICROCODE ALSO HANDLES ALL WCS SUPPORT NEEDED BY THE
BASE MACHINE TO PERFORM ITS FUNCTIONS.   THE FOLLOWING IS
A LIST OF THESE ENTRY POINTS AND THEIR FUNCTIONS:

TMS ADDRESS   FUNCTION

0001   USED BY WCSINIT FLOW.   USED TO SET ADDRESS
                     REGISTER TO ZERO AND ALSO WRITES ZERO IN
                     THE WORD.
                     IF USED BY WCS CODE THEN LOADS ADDRESS REGISTER
                     WRITES ADDRESS VALUE INTO THAT ADDRESS AND INCREMENTS
                     THE ADDRESS REGISTER BY ONE.

0004   USED BY WCSINIT FLOW.   WRITES A COUNT INTO WCS AND
                     THEN INCREMENTS THE ADDRESS REGISTER.

0010   LOADS WCS ADDRESS REGISTER WITH VALUE AND THEN WRITES
                     DATA INTO THIS ADDRESS.

0020   LOADS WCS ADDRESS REGISTER WITH VALUE.
                     (BASE MACHINE ALSO SAVES THIS SAME VALUE IN THE A
                     SCRATCHPAD 21).   THIS ROUTINE ALSO PUMPS ONTO BUSDIN
                     THE DATA FROM THIS LOCATION.

0030   USED BY FIRST WORD IN ROUTINE THAT READS WCS STATUS.
                     NOTE THAT THE WCS STATUS IS NOW READ BY THE UCON
                     INTERFACE.   THIS WORD CAN PROBABLY BE REMOVED FROM
                     BASE MACHINE AND THIS ROUTINE FROM THE TMS ROM.

0040   NOT REFERENCED BY THE BASE MACHINE.

THE WCS USER CAN ALSO USE THESE ROUTINES IN THE TMS ROM.   THE
FOLLOWING IS A LIST OF FUNCTIONS WANTED BY WCS USERS AND WAYS
TO REALIZE THESE FUNCTIONS.

E-1

```
TMS ADDRESS          FUNCTION

                     (READ)
0032                 READ DATA ONTO BUSDIN POINTED TO BY THE ADDRESS
                     REGISTER.

                     (READANDINC)
0024                 READ DATA POINTED TO INTO MD AND THEN INCREMENT
                     THE ADDRESS REGISTER.

                     (LOADANDREAD)
0020                 LOAD THE ADDRESS REGISTER WITH VALUE AND READ DATA
                     OUT ONTO BUSDIN.

                     (LOADREADINC)
0034                 LOAD THE ADDRESS REGISTER AND READ THE DATA
                     POINTED TO INTO MD.   FOLLOWS THIS BY INCREMENTING
                     THE ADDRESS REGISTER.

                     (WRITE)
0014                 WRITE DATA INTO THE ADDRESS POINTED TO.

                     (WRITEANDINC)
0004                 WRITE DATA INTO THE LOCATION POINTED TO AND THEN
                     INCREMENT THE ADDRESS REGISTER.

                     (LOADANDWRITE)
0010                 LOAD ADDRESS REGISTER AND WRITE DATA INTO THIS
                     LOCATION.   METHOD OF USING THIS ROUTINE IS:
                     1ST INST.  CLOCK D AT P2 WITH ADDRESS
                     2ND INST.  SET TMSPTR TO 10
                     3RD INST.  CLOCK D AT P2 WITH DATA TO BE WRITTEN
                     4TH INST.  (DATA IS WRITTEN INTO LOCAL STORE)

                     (LOADWRITEINC)
0001                 LOAD THE ADDRESS REGISTER AND WRITE DATA INTO THIS
                     LOCATION.  FOLLOWS THIS BY INCREMENTING THE ADDRESS
                     REGISTER.   (NOTE THAT THIS ROUTINE IS INVOKED
                     IN THE SAME MANNER AS ABOVE METHOD.)

                     (INCANDREAD)
0005                 INCREMENT THE ADDRESS REGISTER AND READ THE DATA OUT ON
                     THE BUSDIN.   DATA IS AVAILABLE AT P3 OF THE SECOND
                     NULL WORD.

                     (LOADADDRESS)
0040                 LOAD THE ADDRESS REGISTER WITH AN ADDRESS.   NOTE THIS
                     FUNCTION IS EQUIVALENT TO LOADANDREAD.

        .TITLE   TMSROM
.IDENT   /V001/
.RADIX   8                          !OCTAL NUMERICS.
.WIDTH   32R                        !DEFINE 32 BIT WORD.
.OBJECT  <19:0>                     !ACTUAL ROM IS 16 BITS FOR TMS
                                    !AND 4 BITS FOR CROM.
.BOUNDS  [0:777]                    !DEFINE 512 WORDS.
.ADDRESS DUM::=<31:23>              !THIS FIELD EXISTS ONLY TO SATISFY THE
                                    !OF THE MICRO ASSEMBLER.
```

E-2

```
        .TOC    TMS MICROWORD FIELD DEFINITIONS.


.FIELD  ALU ::= <0>                             ;BIT IN TMS ROM TO HANDLE ALU FIELD.

        A::= 1                                  ;SND A STRAIGHT THROUGH ALU.
        B ::=0                                  ;SEND B STRAIGHT THROUGH ALU.
.FIELD  BEN ::= <1>'<2>                         ;BUS BIN SOURCE.

        BSPLO ::= 0                             ;LOW HALF OF B SCRATCHPAD.
        BSPHI ::= 1                             ;HIGH HALF OF B SCRATCHPAD.
        CSP ::= 2                               ;C SCRATCHPAD LOCATION.
        BASCON::= 3                             ;BASE CONSTANTS IN THE CSP.

.FIELD  BSEL ::= <3>                            ;B BUS SELECTION.

        MD ::= 0                                ;WHEN ADDRESSING THE BASE CONSTANTS.

.FIELD  AEN ::= <4>                             ;BUS AIN SOURCE.

        ASPLO ::= 0                             ;LOW HALF OF A SCRATCHPAD.
        ASPHI ::= 1                             ;HIGH HALF OF A SCRATCHPAD.

.FIELD  BADDR ::= <7>'<8>'<3>                   ;LOW ORDER THREE BITS OF B ADDRESS.

.FIELD  AADDR ::= <7>'<8>'<5>                   ;LOW ORDER THREE BITS OF A ADDRESS

.FIELD  ALLAADDR ::= <4>'<6>'<7>'<8>'<5>            ;A ADDRESS INTO A SCRATCHPAD.

        R00 ::= 10
        R01 ::= 11
        R02 ::= 12
        R03 ::= 13
        R04 ::= 14
        R05 ::= 15
        R06 ::= 16
        R07 ::= 17
        R10 ::= 0
        R11 ::= 1
        R12 ::= 2
        R13 ::= 3
        R14 ::= 4
        R15 ::= 5
        R16 ::= 6
        R17 ::= 7
        R20 ::= 30
        R21 ::= 31
        R22 ::= 32
        R23 ::= 33
        R24 ::= 34
        R25 ::= 35
        R26 ::= 36
        R27 ::= 37
        R30 ::= 20
        R31 ::= 21
        R32 ::= 22
        R33 ::= 23
        R34 ::= 24
  ,     R35 ::= 25
        R36 ::= 26
        R37 ::= 27                    E-3
```

```
.FIELD   ALLBADDR  ::= <2>'<6>'<7>'<8>'<3>

         R00  ::=  10
         R01  ::=  11
         R02  ::=  12
         R03  ::=  13
         R04  ::=  14
         R05  ::=  15
         R06  ::=  16
         R07  ::=  17
         R10  ::=  0
         R11  ::=  1
         R12  ::=  2
         R13  ::=  3
         R14  ::=  4
         R15  ::=  5
         R16  ::=  6
         R17  ::=  7
         R20  ::=  30
         R21  ::=  31
         R22  ::=  32
         R23  ::=  33
         R24  ::=  34
         R25  ::=  35
         R26  ::=  36
         R27  ::=  37
         R30  ::=  20
         R31  ::=  21
         R32  ::=  22
         R33  ::=  23
         R34  ::=  24
         R35  ::=  25
         R36  ::=  26
         R37  ::=  27

.FIELD   ALLAADDRW  ::= <12>'<6>'<7>'<8>'<5>
         R00  ::=  10
         R01  ::=  11
         R02  ::=  12
         R03  ::=  13
         R04  ::=  14
         R05  ::=  15
         R06  ::=  16
         R07  ::=  17
         R10  ::=  0
         R11  ::=  1
         R12  ::=  2
         R13  ::=  3
         R14  ::=  4
         R15  ::=  5
         R16  ::=  6
         R17  ::=  7
         R20  ::=  30
         R21  ::=  31
         R22  ::=  32
         R23  ::=  33
         R24  ::=  34
         R25  ::=  35
         R26  ::=  36
```

E-4

```
                R27 ;:= 37
                R30 ;:= 20
                R31 ;:= 21
                R32 ;:= 22
                R33 ;:= 23
                R34 ;:= 24
                R35 ;:= 25
                R36 ;:= 26
                R37 ;:= 27

    .FIELD   FLTPT ;:= <6>                   !A/B SCRATCHPAD ADDRESS BIT 3.

             YES ;:= 0                        !THIS BIT IS ASSERTED LOW AND
             NO  ;:= 1                        !NOT HIGH AS NORMAL.

    .FIELD   CSPADR ;:= <6>'<7>'<8>'<10>     !ADDRESS WHEN ACCESSING C SCRATCHPAD.

    .FIELD   CLKD ;:= <9>                     !CLOCK D REGISTER.

             YES ;:= 1

    .FIELD   WRCSP ;:= <11>                   !WRITE C SCRATCHPAD.

             YES ;:= 1                        !

    .FIELD   HILO ;:= <12>                    !WHEN WRITING TO A OR B SCRATCHPADS
                                              !THIS FIELD SPECIFIES HIGH OR LOW HALF.

             HI ;:= 1                          !HIGH HALF.
             LO ;:= 0                          !LOW HALF.

    .FIELD   WRSP ;:=<14>'<15>                !WRITE BSP,ASP, OR BOTH.

             A ;:= 1                           !WRITE A SCRATCHPAD.
             B ;:= 2                           !WRITE B SCPATCHBAD.
             BOTH ;:=3                         !WRITE BOTH A AND B SCRATCHPADS

    .FIELD   UADRLD ;:= <16>,1                !CROM 0 LOAD ADDRESS REGISTER.

             YES ;:=0                          !LOAD ADDRESS REGISTER.

    .FIELD   UADRIC ;:= <17>                  !CROM 1 INCREMENT ADDRESS REGISTER.

             YES ;:= 1                         !INCREMENT ADDRESS REGISTER.

    .FIELD   INCTMS ;:= <18>,1                !DEFAULT IS TO INCREMENT TMS POINTER.

             YES ;:= 1                         !INCREMENT TMS POINTER.
             NO ;:= 0                          !END OF TMS ROUTINE.

    .FIELD   UWRITE ;:= <19>                  !READ/WRITE LOCAL STORE.

             YES ;:= 1                         !WRITE INTO LOCAL STORE.


             .TOC    TMS MACRO DEFINITIONS
```

```
!           MOVE DATA INTO,AROUND AND OUT OF THF DATAPATH.


.MACRO      MD.DATA  ::=                 BEN/BASCON,BSEL/MD,WRCSP/YES
.MACRO      D.MD.DATA ::=                MD.DATA,CLKD/YES
.MACRO      D.MD    ::=                  BEN/BASCON,BSEL/MD,ALU/B,CLKD/YES


!           CROM MACROS

.MACRO      LOADANDWRITE ::=             UADRLD/YES,UWRITE/YES
.MACRO      POSTINCADDR ::=              UADRIC/YES
.MACRO      GOODBYE  ::=                 INCTMS/0
.MACRO      WRITEDATA ::=                UWRITE/YES
.MACRO      FILLER  ::=                  INCTMS/0
.MACRO      LOADADDRESS ::=              UADRLD/YES
.MACRO      INCANDWRITE ::=              UADRIC/YES,UWRITE/YES
.MACRO      INCADDRESS ::=               UADRIC/YFS
.MACRO      STEPTHROUGH ::=              INCTMS/1


!           GENERAL REGISTER MACROS

.MACRO      LOADGRT  ::=                 D.MD.DATA,WRSP/BOTH,FLTPT/NO
.MACRO      LOADGR(N) ::=                LOADGRT,AADDR/#N
.MACRO      STOREGR(N) ::=               ALU/A,FLTPT/NO,AADDR/#N,CLKD/YES
.MACRO      LOADUSERR6 ::=               D.MD.DATA,WRSP/BOTH,FLTPT/YES,AADDR/6
.MACRO      STOREUSERR6 ::=              ALU/A,FLTPT/YES,AADDR/6,CLKD/YES


!           FLOATING POINT REGISTERS IN A AND B SCRATCHPADS MACROS.

.MACRO      LDFP3(N) ::=                 D.MD.DATA,WRSP/A,AADDR/#N,FLTPT/YES
.MACRO      LDFP2(N) ::=                 D.MD.DATA,WRSP/B,AADDR/#N,FLTPT/YES
.MACRO      LDFP1(N) ::=                 LDFP3(#N),HILO/HI
.MACRO      LDFP0(N) ::=                 LDFP2(#N),HILO/HI
.MACRO      STFP3(N) ::=                 ALU/A,AADDR/#N,FLTPT/YES,CLKD/YES
.MACRO      STFP2(N) ::=                 ALU/B,BADDR/#N,FLTPT/YES,CLKD/YES
.MACRO      STFP1(N) ::=                 STFP3(#N),AEN/ASPHI
.MACRO      STFP0(N) ::=                 STFP2(#N),BEN/BSPHI


!           WCS SCRATCHPAD REGISTER MACROS

.MACRO      LDWCSA(N) ::=                D.MD.DATA,HILO/HI,WRSP/A,FLTPT/NO,AADDR/#N
.MACRO      LDWCSB(N) ::=                D.MD.DATA,HILO/HI,WRSP/B,FLTPT/NO,AADDR/#N
.MACRO      STWCSA(N) ::=                ALU/A,AEN/ASPHI,CLKD/YES,FLTPT/NO,AADDR/#N
.MACRO      STWCSB(N) ::=                ALU/B,BEN/BSPHI,CLKD/YES,FLTPT/NO,BADDR/#N


!           C SCRATCHPAD REGISTER MACROS

.MACRO      LDCSP(N) ::=                 BEN/CSP,WRCSP/YES,CSPADR/#N
.MACRO      STCSP(N) ::=                 ALU/B,BEN/CSP,CLKD/YES,CSPADR/#N


!           ALL SCRATCHPAD MACROS

.MACRO      LOADASPAD(N) ::=             D.MD.DATA,WRSP/A,ALLAADDRW/#N
.MACRO      LOADBSPAD(N) ::=             D.MD.DATA,WRSP/B,ALLAADDRW/#N
.MACRO      STOREASPAD(N) ::=            ALU/A,CLKD/YES,ALLAADDR/#N
.MACRO      STOREBSPAD(N) ::=            ALU/B,CLKD/YES,ALLBADDR/#N
.MACRO      LOADALLCSP(N) ::=            BEN/CSP,WRCSP/YES,CSPADR/#N
.MACRO      STOREALLCSP(N) ::=           ALU/B,BEN/CSP,CLKD/YES,CSPADR/#N
```

E-6

USING ROUTINES IN THE TMS ROM.

THE ROUTINES IN THE TMS ROM ARE DESIGNED TO SAVE DIFFERENT
SETS OF THE 11/60 MACHINE STATE INTO WCS ACTING AS A LOCAL STORE
AND ALSO TO RESTORE THESE SETS FROM DATA IN THE LOCAL STORE.

THESE ROUTINES ARE DESIGNED FOR OPTIMUM DATA FLOW TO FACILITATE
IMPLEMENTATION OF FUNCTIONS SUCH AS CONTEXT SWITCHING WHICH MUST
HAPPEN AS FAST AS POSSIBLE. BECAUSE OF THIS OTHER USES OF THESE
ROUTINES AND SUBSETS OF THESE ROUTINES MAY NOT BE AS EASY TO USE AS
WOULD BE LIKED.

ALL ROUTINES ARE ENTERED WITH THE WCS LOCAL STORE MINUS ONE (LSADR-1)
CLOCKED INTO D. RETURN TO THE WCS ROUTINE WILL OCCUR AFTER THE
FUNCTION HAS BEEN COMPLETED. THESE ROUTINES ARE IMPLEMENTED
BY SETTING UP A PIPELINE IN THE DATAPATH WHERE TWO DIFFERENT
PARTS OF THE DATAPATH MOVE DURING THE SAME MICROCYCLE. THE PIPELINE
CONTINUES UNTIL ALL DATA IN THIS SET HAS BEEN MOVED.

USING SUBSETS OF THESE ROUTINES TO MOVE ONLY A FEW OF THE DATA
ITEMS AND NOT THE WHOLE SET IS NOT EASY. AS AN EXAMPLE THE
FOLLOWING IS THE PROCEDURE TO SAVE REGISTERS R3-R0:

(1)   USE A ROUTINE TO LOAD THE ADDRESS-2 INTO THE ADDRESS REGISTER.
(2)   CLOCK R3 INTO THE D REGISTER.
(3)   SET THE TMSPTR WITH ADDRESS THAT WRITES R4 INTO THE
        ARRAY AND MOVES R3 THROUGH THE DATAPATH AND CLOCKS IT
        INTO D. ONLY THE CROM BITS ON THIS INSTRUCTION WILL BE
        EXECUTED. THE TMS BITS WILL NOT BE ACCESSED. THIS WILL
        WRITE R3 INTO THE LOCAL STORE ADDRESS-1.
(4)   THE NEXT INSTRUCTION WILL WRITE R3 INTO THE ADDRESS AND
        MOVE R2 INTO D. THE REST OF THE ROUTINE WILL WRITE
        R2-R0 INTO THE ARRAY AND RETURN CONTROL TO THE WCS
        ROUTINE AT THE THIRD INSTRUCTION AFTER THE ONE THAT
        SET THE TMSPTR VALUE.

THIS EXAMPLE SHOWS THAT A SUBSET OF THE DATA ITEMS
CANNOT BE STORED IN THE SAME MANNER AS THE ENTIRE SET SINCE
THE FIRST DATA ITEM WILL BE SAVED TWICE IN TWO DIFFERENT ADDRESSES.

.CODE

```
                                 !LOCATION ZERO OF THE TMS IS ACCESSED
                                 !WHENEVER AN INIT SIGNAL IS RAISED
                                 !BY THE BASE MACHINE.  ADDRESS REGISTER
        FILLER                   !WILL CYCLE HERE UNTIL THE TMSPTR  IS CHANGED.
  0 00000000 10000001 00000000 00000000
TMS001:
        LOADANDWRITE             !LOAD ADDRESS, WRITE DATA AND
  1 00000001 00001100 00000000 00000000
        POSTINCADDR              !THEN INCREMENT TO POINT TO THE
  2 00000001 10000111 00000000 00000000
        GOODBYE                  !NEXT ADDRESS.
  3 00000010 00000001 00000000 00000000
```

```
TMS004:
        WRITEDATA                       !WRITE DATA FROM DOUT INTO WCS
    4   00000010 10001101 00000000 00000000
        POSTINCADDR                     !AND INCREMENT TO POINT TO THE
    5   00000011 00000111 00000000 00000000
        GOODBYE                         !NEXT ADDRESS.
    6   00000011 10000001 00000000 00000000
        FILLER                          !
    7   00000100 00000001 00000000 00000000


TMS010:
        LOADANDWRITE                    !LOAD ADDRESS AND WRITE DATA.
   10   00000100 10001100 00000000 00000000
        GOODBYE                         !
   11   00000101 00000001 00000000 00000000
        FILLER
   12   00000101 10000001 00000000 00000000
        FILLER
   13   00000110 00000001 00000000 00000000
        WRITEDATA                       !WRITE DATA FROM DOUT INTO LOCAL STORE
   14   00000110 10001101 00000000 00000000
        GOODBYE                         !
   15   00000111 00000001 00000000 00000000
        FILLER
   16   00000111 10000001 00000000 00000000
        FILLER
   17   00001000 00000001 00000000 00000000
TMS020:
        LOADADDRESS                     !INITIALIZE ADDRESS REGISTER TO VALUE.
   20   00001000 10000100 00000000 00000000
        GOODBYE
   21   00001001 00000001 00000000 00000000
        FILLER
   22   00001001 10000001 00000000 00000000
        FILLER
   23   00001010 00000001 00000000 00000000
READANDINC:
        STEPTHROUGH                     !DO NOTHING BUT READ DATA
   24   00001010 10000101 00000000 00000000
        MD_DATA,STEPTHROUGH             !READ DATA AND PUT INTO MD
   25   00001011 00000101 00001000 00000110
        INCADDRESS                      !INCREMENT ADDRESS REGISTER.
   26   00001011 10000111 00000000 00000000
        GOODBYE                         !EXIT BACK TO CALLER.
   27   00001100 00000001 00000000 00000000


TMS030: STEPTHROUGH
   30   00001100 10000101 00000000 00000000
        STEPTHROUGH                     !
   31   00001101 00000101 00000000 00000000
        STEPTHROUGH
   32   00001101 10000101 00000000 00000000
        FILLER
   33   00001110 00000001 00000000 00000000
LOADREADANDINC:
        LOADADDRESS                     !LOAD ADDRESS REGISTER.
   34   00001110 10000100 00000000 00000000
        MD_DATA,STEPTHROUGH             !WRITE DATA INTO MD.
•  35   00001111 00000101 00001000 00000110
        INCADDRESS                      !POINT TO NEXT ADDRESS.
   36   00001111 10000111 00000000 00000000
        GOODBYE                         !EXIT BACK TO CALLER.
   37   00010000 00000001 00000000 00000000
```

E-8

```
        LOADADDRESS
 40   00010000  10000100  00000000  00000000
        GOODBYE
 41   00010001  00000001  00000000  00000000
```

        .TOC   LOAD GENERAL REGISTERS FROM WCS

| THE GENERAL REGISTERS R0-R7 AND USER R6 ARE LOADED WITH DATA
| SAVED IN THE WCS LOCAL STORE IN 9 CONTIGUOUS LOCATIONS.   THE WCS CODE
| SHOULD LOAD THE STARTING ADDRESS MINUS ONE (LSADR-1) INTO D AND TRANSF
| CONTROL TO THIS ROUTINE.   THE DATA IS PIPELINED FROM LOCAL STORE TO
| THE GENERAL REGISTERS. (IN EACH CYCLE ONE 16 BIT VALUE IS BEING
| READ OUT OF LOCAL STORE AND WRITTEN INTO THE
| MD REGISTER WHILE A SECOND 16 BIT VALUE IS MOVED FROM MD THROUGH THE
| ALU INTO D AND THEN WRITTEN INTO BOTH THE A AND B SCRATCHPADS.)
| CONTROL IS RETURNED TO THE WCS MICROCODE AT THE END OF THIS
| ROUTINE WHEN CROM<2> GOES LOW.
|
LOADGRS:
        LOADADDRESS
```
 42   00010001  10000100  00000000  00000000
        MD_DATA,INCADDRESS
 43   00010010  00000111  00001000  00000110
        LOADGR(7),INCADDRESS
 44   00010010  10000111  11001011  11100110
        LOADGR(6),INCADDRESS
 45   00010011  00000111  11001011  11000110
        LOADUSERR6,INCADDRESS
 46   00010011  10000111  11001011  10000110
        LOADGR(5),INCADDRESS
 47   00010100  00000111  11001010  11100110
        LOADGR(4),INCADDRESS
 50   00010100  10000111  11001010  11000110
        LOADGR(3),INCADDRESS
 51   00010101  00000111  11001011  01100110
        LOADGR(2),INCADDRESS
 52   00010101  10000111  11001011  01000110
        LOADGR(1),INCADDRESS
 53   00010110  00000111  11001010  01100110
        LOADGR(0)
 54   00010110  10000101  11001010  01000110
        GOODBYE
 55   00010111  00000001  00000000  00000000
        FILLER
 56   00010111  10000001  00000000  00000000
        FILLER
 57   00011000  00000001  00000000  00000000
```

.TOC   STORE GENERAL REGISTERS INTO WCS.

!
!   THE GENERAL REGISTERS R0-R7 AND USER R6 ARE SAVED INTO THE WCS LOCAL
!   STORE IN 9 CONTIGUOUS LOCATIONS.   THIS CODE IS INITIATED BY LOADING
!   TMSPTR WITH THE STARTING ADDRESS OF THIS ROUTINE AND CLOCKING
!   INTO D THE LOCAL ADDRESS MINUS ONE (LSADR-1).   THE DATA IS
!   PIPELINED FROM THE A SCRATCHPAD TO WCS LOCAL STORE. (IN EACH CYCLE ONE
!   16 BIT VALUE IS READ OUT OF THE A SCRATCHPAD THROUGH THE ALU AND
!   CLOCKED INTO D WHILE ANOTHER 16 BIT VALUE WHICH HAD BEEN CLOCKED INTO
!   DB PREVIOUSLY IS WRITTEN INTO A LOCAL STORE ADDRESS.)   CONTROL
!   RETURNS TO THE WCS MICROCODE AT THE END OF THE ROUTINE WHEN CROM <2>
!   GOES LOW.
!
STOREGRS:
          LOADADDRESS
   60   00011000 10000100 00000000 00000000
          STOREGR(7)
   61   00011001 00000101 00000011 11100001
          STOREGR(6),INCANDWRITE
   62   00011001 10001111 00000011 11000001
          STOREUSERR6,INCANDWRITE
   63   00011010 00001111 00000011 12000001
          STOREGR(5),INCANDWRITE
   64   00011010 10001111 00000010 11100001
          STOREGR(4),INCANDWRITE
   65   00011011 00001111 00000010 11000001
          STOREGR(3),INCANDWRITE
   66   00011011 10001111 00000011 01100001
          STOREGR(2),INCANDWRITE
   67   00011100 00001111 00000011 01000001
          STOREGR(1),INCANDWRITE
   70   00011100 10001111 00000010 01100001
          STOREGR(0),INCANDWRITE
   71   00011101 00001111 00000010 01000001
          INCANDWRITE
   72   00011101 10001111 00000000 00000000
          GOODBYE
   73   00011110 00000001 00000000 00000000
          FILLER
   74   00011110 10000001 00000000 00000000
          FILLER
   75   00011111 00000001 00000000 00000000

        .TOC   LOAD WARM FLOATING POINT REGISTERS FROM WCS.

THE SIX WARM FLOATING REGISTERS ARE LOADED WITH DATA PREVIOUSLY
SAVED IN THE WCS LOCAL STORE IN 24/10 CONTIGUOUS LOCATIONS.   THE
WCS INITIATES THIS CODE BY LOADING TMSPTR WITH THE STARTING ADDRESS
OF THIS ROUTINE AND CLOCKING INTO D THE LOCAL STORE ADDRESS MINUS ONE
(LSADR-1).   THE DATA IS PIPELINED FROM LOCAL STORE TO EITHER THE A OR
B SCRATCHPADS.   (IN EACH MICROCYCLE ONE 16 BIT VALUE IS BEING READ
OUT OF LOCAL STORE AND WRITTEN INTO THE MD REGISTER WHILE A SECOND 16 BI
MOVED OUT OF THE MD THROUGH THE ALU, THROUGH D AND WRITTEN INTO EITHER 1
A OR B SCRATCHPAD DEPENDING UPON THE SIGNIFICANCE OF THE REGISTER.)
CONTROL RETURNS TO THE WCS ROUTINE WHEN CROM<2> GOES LOW.

OADFP:
        LOADADDRESS
76 00011111 10000100 00000000 00000000
        MD_DATA,INCADDRESS
77 00100000 00000111 00001000 00000110
        LDFP0(5),INCADDRESS
00 00100000 10000111 01011010 10100110
        LDFP1(5),INCADDRESS
01 00100001 00000111 10011010 10100110
        LDFP2(5),INCADDRESS
02 00100001 10000111 01001010 10100110
        LDFP3(5),INCADDRESS
03 00100010 00000111 10001010 10100110
        LDFP0(4),INCADDRESS
04 00100010 10000111 01011010 10000110
        LDFP1(4),INCADDRESS
05 00100011 00000111 10011010 10000110
        LDFP2(4),INCADDRESS
06 00100011 10000111 01001010 10000110
        LDFP3(4),INCADDRESS
07 00100100 00000111 10001010 10000110
        LDFP0(3),INCADDRESS
10 00100100 10000111 01011011 00100110
        LDFP1(3),INCADDRESS
11 00100101 00000111 10011011 00100110
        LDFP2(3),INCADDRESS
12 00100101 10000111 01001011 00100110
        LDFP3(3),INCADDRESS
13 00100110 00000111 10001011 00100110
        LDFP0(2),INCADDRESS
14 00100110 10000111 01011011 00000110
        LDFP1(2),INCADDRESS
15 00100111 00000111 10011011 00000110
        LDFP2(2),INCADDRESS
16 00100111 10000111 01001011 00000110
        LDFP3(2),INCADDRESS
17 00101000 00000111 10001011 00000110
        LDFP0(1),INCADDRESS
20 00101000 10000111 01011010 00100110
        LDFP1(1),INCADDRESS
21 00101001 00000111 10011010 00100110
        LDFP2(1),INCADDRESS
22 00101001 10000111 01001010 00100110
        LDFP3(1),INCADDRESS
23 00101010 00000111 10001010 00100110
        LDFP0(0),INCADDRESS
24 00101010 10000111 01011010 00000110
        LDFP1(0),INCADDRESS
25 00101011 00000111 10011010 00000110

E-11

```
        LDFP2(0),INCADDRESS
126  00101011 10000111 01001010 00000110
        LDFP3(0)
127  00101100 00000101 10001010 02000110
        GOODBYE
130  00101100 10000001 00000000 00000000
        FILLER
131  00101101 00000001 00000000 00000000
        FILLER
132  00101101 10000001 00000000 00000000


!
!        THE SIX WARM FLOATING POINT REGISTERS ARE SAVED INTO THE WCS LOCAL
!        STORE INTO 24/10 CONTIGUOUS LOCATIONS.   WITHIN EACH OF THE FLOATING
!        POINT REGISTERS WORDS ARE SAVED IN SIGNIFICANCE ORDER.   THE WCS ROUT!
!        INITIATES THIS CODE BY LOADING TMSPTR WITH THE STARTING ADDRESS OF TH!
!        ROUTINE AND CLOCKING INTO D THE LOCAL STORE ADDRESS MINUS ONE (LSADR-
!        THE DATA IS PIPELINED FROM THE A OR B SCRATCHPADS INTO THE WCS LOCAL
!        STORE.  (IN EACH MICROCYCLE ONE 16 BIT VALUE IS READ OUT OF THE
!        A OR B SCRATCHPAD THROUGH THE ALU AND CLOCKED INTO D WHILE ANOTHER 16
!        VALUE WHICH HAD BEEN CLOCKED INTO DB PREVIOUSLY IS WRITTEN INTO A
!        LOCAL STORE ADDRESS).   CONTROL RETURNS TO THE WCS MICROCODE AT THE
!        END OF THE ROUTINE WHEN CROM<2> GOES LOW.
!
STOREFP:
        LOADADDRESS
133  00101110 00000100 00000000 00000000
        STFP0(5)
134  00101110 10000101 00000010 10001100
        STFP1(5),INCANDWRITE
135  00101111 00001111 00000010 10110001
        STFP2(5),INCANDWRITE
136  00101111 10001111 00000010 10001000
        STFP3(5),INCANDWRITE
137  00110000 00001111 00000010 10100001
        STFP0(4),INCANDWRITE
140  00110000 10001111 00000010 10000100
        STFP1(4),INCANDWRITE
141  00110001 00001111 00000010 10010001
        STFP2(4),INCANDWRITE
142  00110001 10001111 00000010 10000000
        STFP3(4),INCANDWRITE
143  00110010 00001111 00000010 10000001
        STFP0(3),INCANDWRITE
144  00110010 10001111 00000011 00001100
        STFP1(3),INCANDWRITE
145  00110011 00001111 00000011 00110001
        STFP2(3),INCANDWRITE
146  00110011 10001111 00000011 00001000
        STFP3(3),INCANDWRITE
147  00110100 00001111 00000011 00100001
        STFP0(2),INCANDWRITE
150  00110100 10001111 00000011 00000100
        STFP1(2),INCANDWRITE
151  00110101 00001111 00000011 00010001
        STFP2(2),INCANDWRITE
152  00110101 10001111 00000011 00000000
        STFP3(2),INCANDWRITE
153  00110110 00001111 00000011 00000001
```

E-12

```
            STFP0(1),INCANDWRITE
154   00110110 10001111 00000010 00001100
            STFP1(1),INCANDWRITE
155   00110111 00001111 00000010 00110001
            STFP2(1),INCANDWRITE
156   00110111 10001111 00000010 00001000
            STFP3(1),INCANDWRITE
157   00111000 00001111 00000010 00100001
            STFP0(0),INCANDWRITE
160   00111000 10001111 00000010 00000100
            STFP1(0),INCANDWRITE
161   00111001 00001111 00000010 00010001
            STFP2(0),INCANDWRITE
162   00111001 10001111 00000010 00000000
            STFP3(0),INCANDWRITE
163   00111010 00001111 00000010 00000001
            INCANDWRITE
164   00111010 10001111 00000000 00000000
            GOODBYE
165   00111011 00000001 00000000 00000000
            FILLER
166   00111011 10000001 00000000 00000000
            FILLER
167   00111100 00000001 00000000 00000000
```

.TOC   LOAD C SCRATCHPAD EXCEPT BASE CONSTANTS FROM WCS.

| THIS ROUTINE LOADS ALL OF THE ENTRIES IN THE C SCRATCHPAD EXCEPT
| FOR THE BASE CONSTANTS FROM DATA PREVIOUSLY SAVED IN THE WCS LOCAL
| STORE IN 12/10 CONTIGUOUS LOCATIONS.   THIS CODE IS INITIATED BY
| LOADING TMSPTR WITH THE STARTING ADDRESS OF THIS ROUTINE AND CLOCKING
| INTO D THE LOCAL STORE ADDRESS MINUS ONE (LSADR-1).   THE DATA IS
| READ OUT OF WCS LOCAL STORE AND WRITTEN INTO THE C SCRATCHPAD IN ONE
| MICROCYCLE.   EACH CYCLE HANDLES ONE 16 BIT ITEM.   CONTROL RETURNS TO
| THE WCS ROUTINE WHEN CROM<2> GOES LOW.

.OADCSP:
            LOADADDRESS
.70   00111100 10000100 00000000 00000000
            LDCSP(17),INCADDRESS
.71   00111101 00000111 00001101 11000010
            LDCSP(16),INCADDRESS
.72   00111101 10000111 00001001 11000010
            LDCSP(15),INCADDRESS
73   00111110 00000111 00001100 11000010
            LDCSP(14),INCADDRESS
74   00111110 10000111 00001000 11000010
            LDCSP(13),INCADDRESS
75   00111111 00000111 00001101 01000010
            LDCSP(12),INCADDRESS
76   00111111 10000111 00001001 01000010
            LDCSP(11),INCADDRESS
77   01000000 00000111 00001100 01000010
            LDCSP(10),INCADDRESS
00   01000000 10000111 00001000 01000010
            LDCSP(7),INCADDRESS
01   01000001 00000111 00001101 10000010
```

E-13

```
          LDCSP(6),INCADDRESS
202   01000001 10000111 00001001 10000010
          LDCSP(5),INCADDRESS
203   01000010 00000111 00001100 10000010
          LDCSP(4),INCADDRESS
204   01000010 10000111 00001000 10000010
          GOODBYE
205   01000011 00000001 00000000 00000000
          FILLER
206   01000011 10000001 00000000 00000000
          FILLER
207   01000100 00000001 00000000 00000000
```

.TOC   STORE C SCRATCHPAD EXCEPT BASE CONSTANTS INTO WCS.

```
|
|
|         THE C SCRATCHPAD ENTRIES EXCEPT FOR THE BASE CONSTANTS ARE SAVED INTO
|         THE WCS LOCAL STORE INTO 12/10 CONTIGUOUS LOCATIONS.   THIS CODE IS
|         INITIATED BY LOADING TMSPTR WITH THE STARTING ADDRESS OF THIS ROUTINE A
|         CLOCKING INTO D THE LOCAL STORE ADDRESS MINUS ONE (LSADR-1).   THE DATA
|         IS PIPELINED FROM THE C SCRATCHPAD INTO THE WCS LOCAL STORE.   (IN EACH
|         CYCLE ONE 16 BIT VALUE IS READ OUT OF THE C SCRATCHPAD THROUGH THE ALU
|         AND CLOCKED INTO D WHILE ANOTHER 16 BIT VALUE WHICH HAD BEEN
|         PREVIOUSLY CLOCKED INTO DB IS WRITTEN INTO A LOCAL STORE ADDRESS).
|         CONTROL RETURNS TO THE WCS MICROCODE AT THE END OF THE ROUTINE
|         WHEN CROM<2> GOES LOW.
|
STORECSP:
          LOADADDRESS
210   01000100 10000100 00000000 00000000
          STCSP(17)
211   01000101 00000101 00000111 11000010
          STCSP(16),INCANDWRITE
212   01000101 10001111 00000011 11000010
          STCSP(15),INCANDWRITE
213   01000110 00001111 00000110 11000010
          STCSP(14),INCANDWRITE
214   01000110 10001111 00000010 11000010
          STCSP(13),INCANDWRITE
215   01000111 00001111 00000111 01000010
          STCSP(12),INCANDWRITE
216   01000111 10001111 00000011 01000010
          STCSP(11),INCANDWRITE
217   01001000 00001111 00000110 01000010
          STCSP(10),INCANDWRITE
220   01001000 10001111 00000010 01000010
          STCSP(7),INCANDWRITE
221   01001001 00001111 00000111 10000010
          STCSP(6),INCANDWRITE
222   01001001 10001111 00000011 10000010
          STCSP(5),INCANDWRITE
223   01001010 00001111 00000110 10000010
          STCSP(4),INCANDWRITE
224   01001010 10001111 00000010 10000010
          INCANDWRITE
225   01001011 00001111 00000000 00000000

          GOODBYE
*226  01001011 10000001 00000000 00000000
          FILLER
227   01001100 00000001 00000000 00000000
```

E-14

```
          FILLER
230   01001100 10000001 00000000 00000000

.TOC    LOAD WCS USER SCRATCH REGISTERS FROM WCS.

LOADWCSAB:
          LOADADDRESS
231   01001101 00000100 00000000 00000000
          MD_DATA,INCADDRESS
232   01001101 10000111 00001000 00000110
          LDWCSA(0),INCADDRESS
233   01001110 00000111 10011010 01000110
          LDWCSB(0),INCADDRESS
234   01001110 10000111 01011010 01000110
          LDWCSB(1)
235   01001111 00000101 01011010 01100110
          GOODBYE
236   01001111 10000001 00000000 00000000
          FILLER
237   01010000 00000001 00000000 00000000
          FILLER
240   01010000 10000001 00000000 00000000

.TOC    STORE WCS USER SCRATCH REGISTERS FROM WCS.

STOREWCSAB:
          LOADADDRESS
241   01010001 00000100 00000000 00000000
          STWCSA(0)
242   01010001 10000101 00000010 01010001
          STWCSB(0),INCANDWRITE
243   01010010 00001111 00000010 01000100
          STWCSB(1),INCANDWRITE
244   01010010 10001111 00000010 01001100
          INCANDWRITE
245   01010011 00001111 00000000 00000000
          GOODBYE
246   01010011 10000001 00000000 00000000
          FILLER
247   01010100 00000001 00000000 00000000
          FILLER
250   01010100 10000001 00000000 00000000

.TOC    TMS UTILITY ROUTINES TO SET LOCAL STORE ADDRESS

SETLOAD:
          LOADADDRESS
251   01010101 00000100 00000000 00000000
          MD_DATA,INCADDRESS
252   01010101 10000111 00001000 00000110
          GOODBYE
253   01010110 00000001 00000000 00000000

SETSTORE:
          LOADADDRESS
254   01010110 10000100 00000000 00000000
          GOODBYE
255   01010111 00000001 00000000 00000000
```

.TOC   LOAD ALL A SCRATCHPAD VALUES.

|
| ALL OF THE REGISTERS IN THE A SCRATCHPAD ARE LOADED WITH DATA WHICH
| COMES FROM THE WCS LOCAL STORE IN 32 CONTIGUOUS LOCATIONS.   THE
| WCS CODE SHOULD LOAD THE STARTING ADDRESS MINUS ONE (LSADR-1)
| INTO D AND TRANSFER CONTROL TO THIS ROUTINE.   THIS ROUTINE WILL
| PIPELINE THE DATA FROM LOCAL STORE TO THE GENERAL REGISTERS.
| (IN EACH MICROCYCLE ONE 16 BIT VALUE IS READ OUT OF LOCAL STORE
| AND WRITTEN INTO THE MD REGISTER WHILE A SECOND 16 BIT
| VALUE IS MOVED FROM MD THROUGH THE ALU INTO D AND THEN WRITTEN
| INTO THE A SCRATCHPADS.)   CONTROL IS RETURNED TO THE
| WCS MICROCODE AT THE END OF THIS ROUTINE WHEN CROM<2> GOES LOW.
|
ASPADLOAD:
        LOADADDRESS
256   01010111 10000100 00000000 00000000
        MD_DATA,INCADDRESS
257   01011000 00000111 00001000 00000110
        LOADASPAD(R00),INCADDRESS
260   01011000 10000111 10001010 01000110
        LOADASPAD(R01),INCADDRESS
261   01011001 00000111 10001010 01100110
        LOADASPAD(R02),INCADDRESS
262   01011001 10000111 10001011 01000110
        LOADASPAD(R03),INCADDRESS
263   01011010 00000111 10001011 01100110
        LOADASPAD(R04),INCADDRESS
264   01011010 10000111 10001010 11000110
        LOADASPAD(R05),INCADDRESS
265   01011011 00000111 10001010 11100110
        LOADASPAD(R06),INCADDRESS
266   01011011 10000111 10001011 11000110
        LOADASPAD(R07),INCADDRESS
267   01011100 00000111 10001011 11100110
        LOADASPAD(R10),INCADDRESS
270   01011100 10000111 10001010 00000110
        LOADASPAD(R11),INCADDRESS
271   01011101 00000111 10001010 00100110
        LOADASPAD(R12),INCADDRESS
272   01011101 10000111 10001011 00000110
        LOADASPAD(R13),INCADDRESS
273   01011110 00000111 10001011 00100110
        LOADASPAD(R14),INCADDRESS
274   01011110 10000111 10001010 10000110
        LOADASPAD(R15),INCADDRESS
275   01011111 00000111 10001010 10100110
        LOADASPAD(R16),INCADDRESS
276   01011111 10000111 10001011 10000110
        LOADASPAD(R17),INCADDRESS
277   01100000 00000111 10001011 10100110
        LOADASPAD(R20),INCADDRESS
300   01100000 10000111 10011010 01000110
        LOADASPAD(R21),INCADDRESS
301   01100001 00000111 10011010 01100110
        LOADASPAD(R22),INCADDRESS
302   01100001 10000111 10011011 01000110
        LOADASPAD(R23),INCADDRESS
• 303   01100010 00000111 10011011 01100110

```
        LOADASPAD(R24),INCADDRESS
304  01100010 10000111 10011010 11000110
        LOADASPAD(R25),INCADDRESS
305  01100011 00000111 10011010 11100110
        LOADASPAD(R26),INCADDRESS
306  01100011 10000111 10011011 11000110
        LOADASPAD(R27),INCADDRESS
307  01100100 00000111 10011011 11100110
        LOADASPAD(R30),INCADDRESS
310  01100100 10000111 10011010 00000110
        LOADASPAD(R31),INCADDRESS
311  01100101 00000111 10011010 00100110
        LOADASPAD(R32),INCADDRESS
312  01100101 10000111 10011011 00000110
        LOADASPAD(R33),INCADDRESS
313  01100110 00000111 10011011 00100110
        LOADASPAD(R34),INCADDRESS
314  01100110 10000111 10011010 10000110
        LOADASPAD(R35),INCADDRESS
315  01100111 00000111 10011010 10100110
        LOADASPAD(R36),INCADDRESS
316  01100111 10000111 10011011 10000110
        LOADASPAD(R37)
317  01101000 00000101 10011011 10100110
        GOODBYE
320  01101000 10000001 00000000 00000000
        FILLER
321  01101001 00000001 00000000 00000000
        FILLER
322  01101001 10000001 00000000 00000000


.TOC    STORE ALL A SCRATCHPAD VALUES INTO WCS LOCAL STORE.

!
!       ALL OF THE A SCRATCHPAD REGISTERS ARE SAVED INTO THE WCS
!       LOCAL STORE IN 32 CONTIGUOUS LOCATIONS.   THE WCS INITIATES
!       THIS CODE BY LOADING TMSPTR WITH THE STARTING ADDRESS OF
!       THIS ROUTINE AND CLOCKING INTO D THE LOCAL STORE ADDRESS
!       MINUS ONE (LSADR-1).   THIS ROUTINE PIPELINES THE DATA
!       FROM THE A SCRATCHPAD TO LOCAL STORE.  ( IN EACH MICROCYCLE
!       ONE 16 BIT VALUE IS READ OUT OF THE A SCRATCHPAD THROUGH THE ALU
!       AND CLOCKED INTO D WHILE ANOTHER 16 BIT VALUE WHICH HAD BEEN
!       CLOCKED INTO DB PREVIOUSLY IS WRITTEN INTO A LOCAL STORE ADDRESS.)
!       CONTROL RETURNS TO THE WCS ROUTINE WHEN CROM<2> GOES LOW.
!

ASPADSTORE:
        LOADADDRESS:
        STOREASPAD(R00)
323  01101010 00000101 00000010 01000001
        STOREASPAD(R01),INCANDWRITE
324  01101010 10001111 00000010 01100001
        STOREASPAD(R02),INCANDWRITE
325  01101011 00001111 00000011 01000001
        STOREASPAD(R03),INCANDWRITE
326  01101011 10001111 00000011 01100001
        STOREASPAD(R04),INCANDWRITE
327  01101100 00001111 00000010 11000001
        STOREASPAD(R05),INCANDWRITE
330  01101100 10001111 00000010 11100001
```

```
        STOREASPAD(R06),INCANDWRITE
331     01101101 00001111 00000011 11000001
        STOREASPAD(R07),INCANDWRITE
332     01101101 10001111 00000011 11100001
        STOREASPAD(R10),INCANDWRITE
333     01101110 00001111 00000010 00000001
        STOREASPAD(R11),INCANDWRITE
334     01101110 10001111 00000010 00100001
        STOREASPAD(R12),INCANDWRITE
335     01101111 00001111 00000011 00000001
        STOREASPAD(R13),INCANDWRITE
336     01101111 10001111 00000011 00100001
        STOREASPAD(R14),INCANDWRITE
337     01110000 00001111 00000010 10000001
        STOREASPAD(R15),INCANDWRITE
340     01110000 10001111 00000010 10100001
        STOREASPAD(R16),INCANDWRITE
341     01110001 00001111 00000011 10000001
        STOREASPAD(R17),INCANDWRITE
342     01110001 10001111 00000011 10100001
        STOREASPAD(R20),INCANDWRITE
343     01110010 00001111 00000010 01010001
        STOREASPAD(R21),INCANDWRITE
344     01110010 10001111 00000010 01110001
        STOREASPAD(R22),INCANDWRITE
345     01110011 00001111 00000011 01010001
        STOREASPAD(R23),INCANDWRITE
346     01110011 10001111 00000011 01110001
        STOREASPAD(R24),INCANDWRITE
347     01110100 00001111 00000010 11010001
        STOREASPAD(R25),INCANDWRITE
350     01110100 10001111 00000010 11110001
        STOREASPAD(R26),INCANDWRITE
351     01110101 00001111 00000011 11010001
        STOREASPAD(R27),INCANDWRITE
352     01110101 10001111 00000011 11110001
        STOREASPAD(R30),INCANDWRITE
353     01110110 00001111 00000010 00010001
        STOREASPAD(R31),INCANDWRITE
354     01110110 10001111 00000010 00110001
        STOREASPAD(R32),INCANDWRITE
355     01110111 00001111 00000011 00010001
        STOREASPAD(R33),INCANDWRITE
356     01110111 10001111 00000011 00110001
        STOREASPAD(R34),INCANDWRITE
357     01111000 00001111 00000010 10010001
        STOREASPAD(R35),INCANDWRITE
360     01111000 10001111 00000010 10110001
        STOREASPAD(R36),INCANDWRITE
361     01111001 00001111 00000011 10010001
        STOREASPAD(R37),INCANDWRITE
362     01111001 10001111 00000011 10110001
        INCANDWRITE
363     01111010 00001111 00000000 00000000
        GOODBYE
364     01111010 10000001 00000000 00000000
        FILLER
365     01111011 00000001 00000000 00000000
        FILLER
366     01111011 10000001 00000000 00000000
```

E-18

```
;
;      ALL OF THE REGISTERS IN THE B SCRATCHPAD ARE LOADED WITH DATA WHICH
;      COMES FROM THE WCS LOCAL STORE IN 32 CONTIGUOUS LOCATIONS.    THE
;      WCS INITIATES THIS CODE BY LOADING TMSPTR WITH THE STARTING ADDRESS
;      OF THIS ROUTINE AND CLOCKING INTO D THE LOCAL STORE ADDRESS MINUS ONE
;      (LSADR-1)    THIS ROUTINE WILL PIPELINE THE DATA FROM LOCAL
;      STORE TO THE GENERAL REGISTERS.   (IN EACH MICROCYCLE ONE
;      16 BIT VALUE IS READ OUT OF LOCAL STORE AND WRITTEN INTO THE MD
;      REGISTER WHILE ANOTHER 16 BIT VALUE IS MOVED FROM MD THROUGH
;      THE ALU INTO D AND THEN WRITTEN INTO THE B SCRATCHPADS).
;      CONTROL IS RETURNED TO THE WCS MICROCODE AT THE END OF THIS
;      ROUTINE WHEN CROM<2> GOES LOW.
;
BSPADLOAD:
        LOADADDRESS
367     01111100 00000100 00000000 00000000
        MD,DATA,INCADDRESS
370     01111100 10000111 00001000 00000110
        LOADBSPAD(R00),INCADDRESS
371     01111101 00000111 01001010 01000110
        LOADBSPAD(R01),INCADDRESS
372     01111101 10000111 01001010 01100110
        LOADBSPAD(R02),INCADDRESS
373     01111110 00000111 01001011 01000110
        LOADBSPAD(R03),INCADDRESS
374     01111110 10000111 01001011 01100110
        LOADBSPAD(R04),INCADDRESS
375     01111111 00000111 01001010 11000110
        LOADBSPAD(R05),INCADDRESS
376     01111111 10000111 01001010 11100110
        LOADBSPAD(R06),INCADDRESS
377     10000000 00000111 01001011 11000110
        LOADBSPAD(R07),INCADDRESS
400     10000000 10000111 01001011 11100110
        LOADBSPAD(R10),INCADDRESS
401     10000001 00000111 01001010 00000110
        LOADBSPAD(R11),INCADDRESS
402     10000001 10000111 01001010 00100110
        LOADBSPAD(R12),INCADDRESS
403     10000010 00000111 01001011 00000110
        LOADBSPAD(R13),INCADDRESS
404     10000010 10000111 01001011 00100110
        LOADBSPAD(R14),INCADDRESS
405     10000011 00000111 01001010 10000110
        LOADBSPAD(R15),INCADDRESS
406     10000011 10000111 01001010 10100110
        LOADBSPAD(R16),INCADDRESS
407     10000100 00000111 01001011 10000110
        LOADBSPAD(R17),INCADDRESS
410     10000100 10000111 01001011 10100110
        LOADBSPAD(R20),INCADDRESS
411     10000101 00000111 01011010 01000110
        LOADBSPAD(R21),INCADDRESS
412     10000101 10000111 01011010 01100110
        LOADBSPAD(R22),INCADDRESS
413     10000110 00000111 01011011 01000110
        LOADBSPAD(R23),INCADDRESS
414     10000110 10000111 01011011 01100110
```

E-19

```
         LOADBSPAD(R24),INCADDRESS
415   10000111 00000111 01011010 11000110
         LOADBSPAD(R25),INCADDRESS
416   10000111 10000111 01011010 11100110
         LOADBSPAD(R26),INCADDRESS
417   10001000 00000111 01011011 11000110
         LOADBSPAD(R27),INCADDRESS
420   10001000 10000111 01011011 11100110
         LOADBSPAD(R30),INCADDRESS
421   10001001 00000111 01011010 00000110
         LOADBSPAD(R31),INCADDRESS
422   10001001 10000111 01011010 00100110
         LOADBSPAD(R32),INCADDRESS
423   10001010 00000111 01011011 00000110
         LOADBSPAD(R33),INCADDRESS
424   10001010 10000111 01011011 00100110
         LOADBSPAD(R34),INCADDRESS
425   10001011 00000111 01011010 10000110
         LOADBSPAD(R35),INCADDRESS
426   10001011 10000111 01011010 10100110
         LOADBSPAD(R36),INCADDRESS
427   10001100 00000111 01011011 10000110
         LOADBSPAD(R37)
430   10001100 10000101 01011011 10100110
         GOODBYE
431   10001101 00000001 00000000 00000000
         FILLER
432   10001101 10000001 00000000 00000000
         FILLER
433   10001110 00000001 00000000 00000000


.TOC     STORE ALL B SCRATCHPAD INTO WCS LOCAL STORE

!
!        ALL OF THE B SCRATCHPAD REGISTERS ARE SAVED INTO THE WCS
!        LOCAL STORE IN 32 CONTIGUOUS LOCATIONS.   THE WCS INITIATES
!        THIS CODE BY LOADING TMSPTR (BY USE OF UCON) WITH THE
!        STARTING ADDRESS OF THIS ROUTINE AND CLOCKING INTO D THE LOCAL
!        STORE ADDRESS MINUS ONE (LSADR-1).   THIS ROUTINE PIPELINES THE
!        ONE 16 BIT VALUE IS READ OUT OF THE B SCRATCHPAD THROUGH THE ALU
!        AND CLOCKED INTO D WHILE ANOTHER 16 BIT VALUE WHICH HAD BEEN
!        CLOCKED INTO DB PREVIOUSLY IS WRITTEN INTO A LOCAL STORE ADDRESS.)
!        CONTROL RETURNS TO THE WCS ROUTINE WHEN CROM<2> GOES LOW.
!
BSPADSTORE:
         LOADADDRESS
434   10001110 10000100 00000000 00000000
         STOREBSPAD(R00)
435   10001111 00000101 00000010 01000000
         STOREBSPAD(R01),INCANDWRITE
436   10001111 10001111 00000010 01001000
         STOREBSPAD(R02),INCANDWRITE
437   10010000 00001111 00000011 01000000
         STOREBSPAD(R03),INCANDWRITE
440   10010000 10001111 00000011 01001000
         STOREBSPAD(R04),INCANDWRITE
441   10010001 00001111 00000010 11000000
         STOREBSPAD(R05),INCANDWRITE
442   10010001 10001111 00000010 11001000
```

E-20

```
              STOREBSPAD(R06),INCANDWRITE
443   10010010 00001111 00000011 11000000
              STOREBSPAD(R07),INCANDWRITE
444   10010010 10001111 00000011 11001000
              STOREBSPAD(R10),INCANDWRITE
445   10010011 00001111 00000010 00000000
              STOREBSPAD(R11),INCANDWRITE
446   10010011 10001111 00000010 00001000
              STOREBSPAD(R12),INCANDWRITE
447   10010100 00001111 00000011 00000000
              STOREBSPAD(R13),INCANDWRITE
450   10010100 10001111 00000011 00001000
              STOREBSPAD(R14),INCANDWRITE
451   10010101 00001111 00000010 10000000
              STOREBSPAD(R15),INCANDWRITE
452   10010101 10001111 00000010 10001000
              STOREBSPAD(R16),INCANDWRITE
453   10010110 00001111 00000011 10000000
              STOREBSPAD(R17),INCANDWRITE
454   10010110 10001111 00000011 10001000
              STOREBSPAD(R20),INCANDWRITE
455   10010111 00001111 00000010 01000100
              STOREBSPAD(R21),INCANDWRITE
456   10010111 10001111 00000010 01001100
              STOREBSPAD(R22),INCANDWRITE
457   10011000 00001111 00000011 01000100
              STOREBSPAD(R23),INCANDWRITE
460   10011000 10001111 00000011 01001100
              STOREBSPAD(R24),INCANDWRITE
461   10011001 00001111 00000010 11000100
              STOREBSPAD(R25),INCANDWRITE
462   10011001 10001111 00000010 11001100
              STOREBSPAD(R26),INCANDWRITE
463   10011010 00001111 00000011 11000100
              STOREBSPAD(R27),INCANDWRITE
464   10011010 10001111 00000011 11001100
              STOREBSPAD(R30),INCANDWRITE
465   10011011 00001111 00000010 00000100
              STOREBSPAD(R31),INCANDWRITE
466   10011011 10001111 00000010 00001100
              STOREBSPAD(R32),INCANDWRITE
467   10011100 00001111 00000011 00000100
              STOREBSPAD(R33),INCANDWRITE
470   10011100 10001111 00000011 00001100
              STOREBSPAD(R34),INCANDWRITE
471   10011101 00001111 00000010 10000100
              STOREBSPAD(R35),INCANDWRITE
472   10011101 10001111 00000010 10001100
              STOREBSPAD(R36),INCANDWRITE
473   10011110 00001111 00000011 10000100
              STOREBSPAD(R37),INCANDWRITE
474   10011110 10001111 00000011 10001100
              INCANDWRITE
475   10011111 00001111 00000000 00000000
              GOODBYE
476   10011111 10000001 00000000 00000000
              FILLER
477   10100000 00000001 00000000 00000000
              FILLER
500   10100000 10000001 00000000 00000000
```

E-21

```
;
;       THIS ROUTINE LOADS ALL OF THE ENTRIES IN THE C SCRATCHPAD FROM DATA
;       PREVIOUSLY SAVED IN THE WCS LOCAL STORE.    THIS CODE IS INITIATED
;       BY LOADING TMSPTR WITH THE STARTING ADDRESS OF THIS ROUTINE AND
;       CLOCKING INTO D THE LOCAL STORE ADDRESS MINUS ONE (LSADP-1).
;       THE DATA IS READ OUT OF WCS LOCAL STORE AND WRITTEN INTO THE
;       C SCRATCHPAD IN ONE MICROCYCLE.    EACH CYCLE HANDLES ONE 16 BIT
;       DATA ITEM.    CONTROL RETURNS TO THE WCS MICROCODE WHEN CROM<2> GOES LO
;
ALLCSPLOAD:
        LOADADDRESS
501     10100001 00000100 00000000 00000000
        LOADALLCSP(0),INCADDRESS
502     10100001 10000111 00001000 00000010
        LOADALLCSP(1),INCADDRESS
503     10100010 00000111 00001100 00000010
        LOADALLCSP(2),INCADDRESS
504     10100010 10000111 00001001 00000010
        LOADALLCSP(3),INCADDRESS
505     10100011 00000111 00001101 00000010
        LOADALLCSP(4),INCADDRESS
506     10100011 10000111 00001000 10000010
        LOADALLCSP(5),INCADDRESS
507     10100100 00000111 00001100 10000010
        LOADALLCSP(6),INCADDRESS
510     10100100 10000111 00001001 10000010
        LOADALLCSP(7),INCADDRESS
511     10100101 00000111 00001101 10000010
        LOADALLCSP(10),INCADDRESS
512     10100101 10000111 00001000 01000010
        LOADALLCSP(11),INCADDRESS
513     10100110 00000111 00001100 01000010
        LOADALLCSP(12),INCADDRESS
514     10100110 10000111 00001001 01000010
        LOADALLCSP(13),INCADDRESS
515     10100111 00000111 00001101 01000010
        LOADALLCSP(14),INCADDRESS
516     10100111 10000111 00001000 11000010
        LOADALLCSP(15),INCADDRESS
517     10101000 00000111 00001100 11000010
        LOADALLCSP(16),INCADDRESS
520     10101000 10000111 00001001 11000010
        LOADALLCSP(17),INCADDRESS
521     10101001 00000111 00001101 11000010
        GOODBYE
522     10101001 10000001 00000000 00000000
        FILLER
523     10101010 00000001 00000000 00000000
        FILLER
524     10101010 10000001 00000000 00000000
```

E-22

ALL THE C SCRATCHPAD ENTRIES ARE SAVED INTO WCS.   THIS ROUTINE IS
CALLED BY LOADING TMSPTR WITH THE STARTING ADDRESS OF THIS ROUTINE AND
CLOCKING INTO D THE C SCRATCHPAD INTO WCS LOCAL STORE.   (IN EACH
MICROCYCLE ONE 16 BIT VALUE IS READ OUT OF THE C SCRATCHPAD, THROUGH
THE ALU AND CLOCKED INTO D WHILE ANOTHER 16 BIT VALUE WHICH HAD
BEEN PREVIOUSLY CLOCKED INTO DB IS WRITTEN INTO A LOCAL STORE ADDRESS.)
CONTROL RETURNS TO THE WCS MICROCODE WHEN CROM<2> GOES LOW AT THE
END OF THIS ROUTINE.

```
.LCSPSTORE:
        LOADADDRESS
!5   10101011 00000100 00000000 00000000
        STOREALLCSP(0)
!6   10101011 10000101 00000010 00000010
        STOREALLCSP(1),INCANDWRITE
!7   10101100 00001111 00000110 00000010
        STOREALLCSP(2),INCANDWRITE
!0   10101100 10001111 00000011 00000010
        STOREALLCSP(3),INCANDWRITE
!1   10101101 00001111 00000111 00000010
        STOREALLCSP(4),INCANDWRITE
!2   10101101 10001111 00000010 10000010
        STOREALLCSP(5),INCANDWRITE
!3   10101110 00001111 00000110 10000010
        STOREALLCSP(6),INCANDWRITE
!4   10101110 10001111 00000011 10000010
        STOREALLCSP(7),INCANDWRITE
!5   10101111 00001111 00000111 10000010
        STOREALLCSP(10),INCANDWRITE
!6   10101111 10001111 00000010 01000010
        STOREALLCSP(11),INCANDWRITE
!7   10110000 00001111 00000110 01000010
        STOREALLCSP(12),INCANDWRITE
!0   10110000 10001111 00000011 01000010
        STOREALLCSP(13),INCANDWRITE
!1   10110001 00001111 00000111 01000010
        STOREALLCSP(14),INCANDWRITE
!2   10110001 10001111 00000010 11000010
        STOREALLCSP(15),INCANDWRITE
!3   10110010 00001111 00000110 11000010
        STOREALLCSP(16),INCANDWRITE
!4   10110010 10001111 00000011 11000010
        STOREALLCSP(17),INCANDWRITE
!5   10110011 00001111 00000111 11000010
        INCANDWRITE
!6   10110011 10001111 00000000 00000000
        GOODBYE
!7   10110100 00000001 00000000 00000000
        FILLER
!0   10110100 10000001 00000000 00000000
        FILLER
!1   10110101 00000001 00000000 00000000
```

E 23

I   THIS ROUTINE REQUIRES A TOTAL OF SIX MICROCYCLES TO LOAD
I   THE ADDRESS REGISTER AND READ TWO PIECES OF DATA FROM LOCAL
I   STORE PLACING THE FIRST IN D AND THE SECOND VALUE IN MD.   THE
I   FOLLOWING DIAGRAM SHOWS HOW TO CALL THIS ROUTINE AND WHAT HAPPENS
I   AT WHAT TIME:

WCS MICROCODE	TMS AND CROM
P2-T, D<---ADDRESS	NO CONTROL
TMSPTR_(LOADREADTWO)	NO CONTROL
NULL WORD	NO CONTROL
P3, MD<---DATA	LOAD ADDRESS (DATA CLOCKED ONTO BUSDIN)
NO CONTROL	INCADDRESS P2-T, D<---MD,(DATA CLOCKED BUS P3, MD<---DATA
NO CONTROL	GOODBYE

I   * BACK IN CONTROL.

LOADREADTWO:
         LOADADDRESS
552   10110101 10000100 00000000 00000000
         D_MD,MD_DATA,INCADDRESS
553   10110110 00000111 00001010 00000110
         GOODBYE
554   10110110 10000001 00000000 00000000
         FILLER
555   10110111 00000001 00000000 00000000

I
I   THIS ROUTINE REQUIRES A TOTAL OF FIVE MICROCYCLES TO READ TWO
I   PIECES OF DATA ASSUMING THE ADDRESS REGISTER HAS ALREADY
I   BEEN LOADED.   THE FOLLOWING DIAGRAM SHOWS HOW TO CALL THIS
I   ROUTINE AND WHAT HAPPENS WHEN:

E-24

```
********************************************************************
*           WCS MICROCODE           *        TMS AND CROM          *
********************************************************************
*                                   *                              *
*       TMSPTR.(INCREADTWO)         *         NO CONTROL           *
*                                   *                              *
********************************************************************
*                                   *                              *
*          NULL WORD                *         NO CONTROL           *
*                                   *                              *
********************************************************************
*                                   *         INCADDRESS          *
*                                   *  (DATA CLOCKED ONTO BUSDIN)  *
*  P3,    MD<-- DATA                *                              *
********************************************************************
*                                   *         INCADDRESS          *
*          NO CONTROL               * P2-T, D<--MD (BUSDIN CLOCKED)*
*                                   * P3,    MD<--DATA             *
********************************************************************
*                                   *                              *
*          NO CONTROL               *          GOODBYE             *
*                                   *                              *
********************************************************************
*BACK IN CONTROL.
```

```
INCREADTWO:
        INCADDRESS
556   10110111 10000111 00000000 00000000
        D.MD,MD.DATA,INCADDRESS
557   10111000 00000111 00001010 00000110
        GOODBYE
560   10111000 10000001 00000000 00000000
```

,TOC   WRITE TWO PIECES OF DATA INTO LOCAL STORE

        THIS ROUTINE REQUIRES A TOTAL OF SIX CYCLES TO LOAD THE
        ADDRESS REGISTER AND WRITE TWO PIECES OF DATA INTO LOCAL
        STORE.   THE FOLLOWING DIAGRAM SHOWS HOW TO CALL THIS
        ROUTINE AND WHAT HAPPENS WHEN:

```
********************************************************************
*           WCS MICROCODE           *        TMS AND CROM          *
********************************************************************
*                                   *                              *
*  P2-T, D<--ADDRESS                *         NO CONTROL           *
*                                   *                              *
********************************************************************
*                                   *      (DB<--ADDRESS)          *
*       TMSPTR-(LOADWRITETWO)        *         NO CONTROL           *
*                                   *                              *
********************************************************************
*                                   *                              *
*  P2-T, D<--DATA1                  *         NO CONTROL           *
*                                   *                              *
```

E-25

```
I     *****************************************************************
I     *                              * LOADADDRESS  (DB<--DATA1)      *
I     * P2-T, D<--DATA2              * (DATA1 WRITTEN INTO LOCAL ST)  *
I     *                              *                                *
I     *****************************************************************
I     *                              * INCADDRESS (DB<--DATA2)        *
I     *      NO CONTROL              * (DATA2 WRITTEN)                *
I     *                              *                                *
I     *****************************************************************
I     *                              *                                *
I     *      NO CONTROL              *          GOODBYE               *
I     *                              *                                *
I     *****************************************************************
I     *BACK IN CONTROL.
```

```
LOADWRITETWO:
        LOADANDWRITE
561   10111001 00001100 00000000 00000000
        INCADDRESS,WRITEDATA
562   10111001 10001111 00000000 00000000
        GOODBYE
563   10111010 00000001 00000000 00000000
        FILLER
564   10111010 10000001 00000000 00000000
```

```
I     THIS ROUTINE REQUIRES A TOTAL OF 5 CYCLES TO WRITE TWO PIECES OF DATA
I     INTO LOCAL STORE ASSUMING THE ADDRESS REGISTER HAS BEEN LOADED
I     PREVIOUSLY.   THE FOLLOWING DIAGRAM SHOWS HOW TO CALL THIS
I     ROUTINE AND WHAT HAPPENS WHEN:
```

```
I     *****************************************************************
I     *      WCS MICROCODE           *        TMS AND CROM            *
I     *****************************************************************
I     *                              *                                *
I     *      TMSPTR (WRITETWO)       *        NO CONTROL             *
I     *                              *                                *
I     *****************************************************************
I     *                              *                                *
I     * P2-T, D<--DATA1              *        NO CONTROL             *
I     *                              *                                *
I     *****************************************************************
I     *                              * INCADDRESS (DB<--DATA1)        *
I     * P2-T, D<DATA2                * (DATA1 WRITTEN)                *
I     *                              *                                *
I     *****************************************************************
I     *                              * INCADDRESS  (DB<--DATA2)       *
I     *      NO CONTROL              * (DATA2 WRITTEN)                *
I     *                              *                                *
I     *****************************************************************
I     *                              *                                *
I     *      NO CONTROL              *          GOODBYE               *
I     *                              *                                *
I     *****************************************************************
I     *BACK IN CONTROL.
```

E-26

```
WRITETWO:
        INCADDRESS,WRITEDATA
565     10111011 00001111 00000000 00000000
        INCADDRESS,WRITEDATA
566     10111011 10001111 00000000 00000000
        GOODBYE
567     10111100 00000001 00000000 00000000
        FILLER
570     10111100 10000001 00000000 00000000
```

.TOC   READ DATA VALUE POINTED TO BY LOCAL STORE VALUE POINTER.

        THIS ROUTINE TAKES THE ADDRESS LOADED, READS THE VALUE IN THAT
        ADDRESS, AND THEN USES THAT VALUE AS AN ADDRESS TO READ THE DATA
        VALUE INTO MD.   THIS ROUTINE REQUIRES SUPPORT FROM THE MICROCODE
        AS DESCRIBED IN THE CALLING SEQUENCE DIAGRAM BELOW:

```
*********************************************************************
*            WCS MICROCODE              *        TMS AND CROM       *
*********************************************************************
*                                       *                           *
* P2=T, D<--ADDRESS                     *        NO CONTROL         *
*                                       *                           *
*********************************************************************
*                                       *     (DB<--ADDRESS)        *
*       TMSPTR (READINDIRECT)           *        NO CONTROL         *
*                                       *                           *
*********************************************************************
*                                       *                           *
*       NULL WORD                       *        NO CONTROL         *
*                                       *                           *
*********************************************************************
*                                       *     LOADADDRESS           *
*                                       *     (CLOCK BUSDIN)        *
* P3,   MD<--ADDRESS                    *                           *
*********************************************************************
*                                       *                           *
*       NO CONTROL                      * P2=T, D<--MD              *
*                                       *                           *
*********************************************************************
*                                       *     (DB<--ADDRESS)        *
*       NO CONTROL                      *                           *
*                                       *                           *
*********************************************************************
*                                       *     LOADADDRESS           *
*       NO CONTROL                      *     (BUSDIN<--DATA)       *
*                                       * P3,   MD<--DATA           *
*********************************************************************
*                                       *                           *
*       NO CONTROL                      *        GOODBYE            *
*                                       *                           *
*********************************************************************
*BACK IN CONTROL.
```

E-27

```
READINDIRECT:
        LOADADDRESS
571   10111101 00000100 00000000 00000000
        D_MD,STEPTHROUGH
572   10111101 10000101 00000010 00000110
        STEPTHROUGH
573   10111110 00000101 00000000 00000000
        LOADADDRESS,MD_DATA
574   10111110 10000100 00001000 00000110
        GOODBYE
575   10111111 00000001 00000000 00000000
        FILLER
576   10111111 10000001 00000000 00000000
```

.TOC   WRITE DATA INTO LOCAL STORE ADDRESS POINTED TO BY LOCAL STORE POINTER.

!
!   THIS ROUTINE USES ADDRESS POINTED TO BY CALLING ROUTINE AS
!   A POINTER TO ANOTHER LOCAL STORE VALUE. THE VALUE IN WCSA(0)
!   IS WRITTEN TO THAT LOCAL STORE POSITION. THE FOLLOWING
!   TIMING DIAGRAM SHOWS HOW THIS ROUTINE IS CALLED AND EXECUTED:


```
!    ********************************************************************
!    *         WCS MICROCODE              *        TMS AND CROM         *
!    ********************************************************************
!    *                                    *                            *
!    * P2-T, D<--ADDRESS                  *        NO CONTROL          *
!    *                                    *                            *
!    ********************************************************************
!    *                                    *   (DB<--ADDRESS)           *
!    *       THSPTR_(WRITEINDIRECT)       *   NO CONTROL               *
!    *                                    *                            *
!    ********************************************************************
!    *                                    *                            *
!    *       NULL WORD                    *   NO CONTROL               *
!    *                                    *                            *
!    ********************************************************************
!    *                                    *        LOAD ADDORESS       *
!    *                                    * (BUSDIN CLOCKED WITH DATA)  *
!    * P3,   MD<--DATA                    *                            *
!    ********************************************************************
!    *                                    *                            *
!    *       NO CONTROL                   * P2-T, D<--MD               *
!    *                                    *                            *
!    ********************************************************************
!    *                                    *   (DB<--ADDRESS)           *
!    *       NO CONTROL                   * P2-T, D<--WCSA(0)          *
!    *                                    *                            *
!    ********************************************************************
!    *                                    * LOADADDRESS (DB<--DATA)    *
!    *       NO CONTROL                   * (DATA WRITTEN)             *
!    *                                    *                            *
!    ********************************************************************
!    *                                    *                            *
!    *       NO CONTROL                   *        GOODBYE             *
!    *                                    *                            *
!    ********************************************************************
!    *BACK IN CONTROL.
```

E-28

```
WRITEINDIRECT:
        LOADADDRESS
577     11000000 00000100 00000000 00000000
        D_MD,STEPTHROUGH
600     11000000 10000101 00000010 00000110
        STWCBA(0)
601     11000001 00000101 00000010 01010001
        LOADADDRESS,WRITEDATA,STEPTHROUGH
602     11000001 10001100 00000000 00000000
        GOODBYE
603     11000010 00000001 00000000 00000000
        FILLER
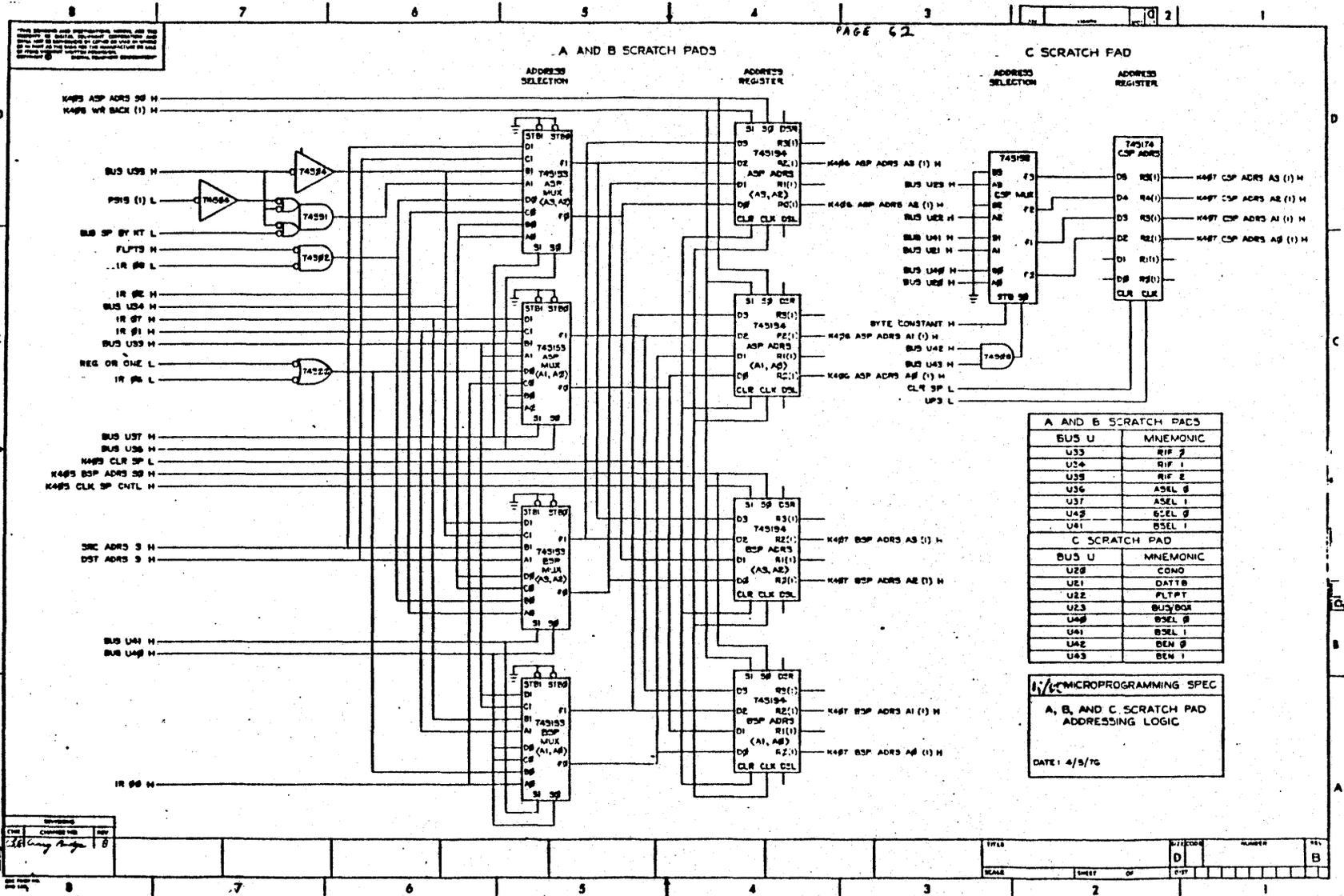604     11000010 10000001 00000000 00000000
```

APPENDIX F
(TEMPORARY)

The following pages contains drawings and other information from the Microprogramming Summary which have not yet been integrated into the specification.

```
FOV=        DOP*SMØ*DMØ*-SR7*-DR7                          1
     +      SOP'*DMØ*-DR7                                  2
     +      DOP*SM(6 OR 7)                                 3
     +      DOP*SMØ*DM(6 OR 7)                             4
     +      SOP*DM(6 OR 7)                                 5

     where SOP' = SOP*-(NEG+SBC+ROR+ASR)
```

used in: 1.   FOV # SERVICE (a BUT)

2.   SP rewrite defeat (both bytes) when -FOV also BUT
(INSTR1)
[if BUT (instr 1) and not fetch overlap then defeat
rewrite].

3.   On a JAMUPP save FOV (sections 1 and 2) in a status flop
readable thru hot box interface or, maybe, used in a BUT.

4.   Conditional DATI in FETØ3
a.   always do DATI
b.   we have a STOP THE BUS line to bus control if not FOV
then yank it (i.e., do stop the "bus").

PAGE 62

F-3

FIGURE 6.21

F-4

FROM FET03

SOURCE ADDRESS CALCULATIONS
AND OPERAND FETCH

SM ≠ ∅

SOURCE ADDRESS
CALCULATION &
DATA ACCESS

TABLES B & C

WORD ≠ DM∅

MD ← DATA
TEST IR <15:12>
GO TO
{ MEM - REG DOP
EXECUTES, TABLE F }

R [DR] ← F (MD, R [DR])

(F)

WORD ≠ -DM∅

MD ← DATA
TEST IR
IF MOV GO TO MOV
EXECUTE ENTRIES
FOR -DM∅
IF -MOV
DO FURTHER DECODE

MOV

(D)

-MOV

R [SRC] ← MD, TEST IR <5:3>
GO TO DESTINATION
ADDRESS CALCULATION
WITH ENTRY POINTS
FOR THE DESTINATION
MODES ≠ ∅

(G)

BYTE ≠ DM∅

MD ← DATA
PLACE ADDRESSED
BYTE OF DATA IN
R [SRC], HI & LO BYTE

TEST IR <15:12>
GO TO
{ MEM - REG DOP
BYTE EXECUTES
TABLE F }

R [DR] <LO> ← F (R[SRC], R[DR])

(F)

BYTE R -DM∅

MD ← DATA, TEST FOR
MOV INSTRUCTION

- MOV

PLACE ADDRESSED
BYTE IN R[SRC]
HI & LO BYTE,
GO TO DESTINATION
ADDRESS CALCULATION
WITH ENTRY POINTS
FOR THE MODES ≠ ∅
TEST IR <5:3>

(∅)

MOV

PLACE ADDRESSED
BYTE IN R [SRC]
HI & LO BYTE
GO TO MOV EXECUTE
FOR CORRESPONDING
DM ≠ ∅

(M)

FIGURE 4.2.2

N.B. : MOV TO REG IS SIGN EXTENDED

F-5

F-6

PAGE 13

FIGURE 6.3.3 (a)

FIGURE 6.2.3 (b)

F-8

FIGURE 6.2.3 (c)

SOURCE

F-9

FIGURE 6.2.3 (4)

TITLE DEST