EK-KB11C-TM-001

KB11-C PROCESSOR MANUAL (PDP-11/70)

digital equipment corporation · maynard. massachusetts

The material in this manual is for informational purposes and is subject to change without notice.

Digital Equipment Corporation assumes no responsibility for any errors which may appear in this manual.

Printed in U.S.A.

This document was set on DIGITAL's DECset-8000 computerized typesetting system.

The following are trademarks of Digital Equipment Corporation, Maynard, Massachusetts:

DEC
FLIP CHIP
DIGITAL
UNIBUS
DECUS
PDP
FOCAL
COMPUTER LAB
MASSBUS

9/85-15

TABLE OF CONTENTS

INTRODUCTION

This manual describes the KB11-C Central Processor Unit, which is the basic component of the PDP-11/70 Programmed Data Processor System. The purpose of this manual is to:

1. provide an overall understanding of how the KB11-C functions in the PDP-11/70 System.
2. describe how the KB11-C logic works in sufficient detail to enable maintenance personnel to perform on-site troubleshooting and repair.

The format of this manual is functional, i.e., the intent is to explain the various processes that are executed by the KB11-C, as opposed to a module by module logic description. Since this might be a problem for a technician who has a module to repair, an index of logic functions by module is provided.

This manual is divided into six sections:

Section I is an introduction to the PDP-11/70. It describes a block diagram of the system and introduces some system concepts.

Section II describes the processor. Its six chapters explain processor control, data manipulation, Control Registers, timing, data transfers and error handling.

Section III provides both an operating guide to the Console and a detailed description of its logic.

Section IV describes Memory Management and address space.

Section V describes the Unibus Map.

Section VI contains a description of the Cache.

Appendix A contains both a System Data Paths and a System Address Paths block diagram.

Due to the numerous references to specific logic functions in the text, it is recommended that the reader refer to the *PDP-11/70 Engineering Print Set* while reading this manual.

Comments (both favorable and unfavorable), suggestions, and corrections are welcome. A Reader's Comment sheet is provided for this purpose at the end of this manual.

RELATED DOCUMENTS

This manual should be used in conjunction with the following related publications:

PDP-11/70 Maintenance and Installation Manual

PDP-11/70 Processor Handbook

MJ11 Memory System Maintenance Manual

FP11-C Floating-Point Processor Manual

RWS04/RWS03 Fixed Head Disk Subsystem Maintenance Manual

RWP04 Moving Head Disk Subsystem Maintenance Manual

TWU16 Magnetic Tape Subsystem Maintenance Manual

PDP-11 Peripherals Handbook

SECTION I

BLOCK DIAGRAM AND CONCEPTS

Unless otherwise indicated, references within this section pertain to this section only.

SECTION I BLOCK DIAGRAM AND CONCEPTS
CONTENTS

ILLUSTRATIONS

CHAPTER 1
BLOCK DIAGRAM

The PDP-11/70 is the most powerful computer in the PDP-11 family. It is designed to operate in large, sophisticated, high-performance systems. It can by used as a powerful computational tool for high-speed, real-time applications and for large multi-user, multi-task, time-shared applications requiring large amounts of addressable memory space. Although it is a 16-bit machine, it applies the power of a Cache memory and 32-bit memory and I/O structure to demanding, multi-function computing requirements.

The PDP-11/70 contains as an integral part of the Central Processor Unit (CPU), the following hardware features and expansion capabilities:

> Cache memory organization to provide bipolar memory speed at core memory prices.
>
> Memory Management for relocation and protection in multi-user, multi-task environments.
>
> Ability to access up to 4 million bytes of Main Memory.
>
> Optional high-speed mass storage controllers as an integral part of the CPU. These controllers provide dedicated paths to high performance storage devices.
>
> Optional Floating Point Processor

1.1 BLOCK DIAGRAM

The PDP-11/70 is a medium scale, general-purpose computer. A block diagram of the computer is shown in Figure 1-1.

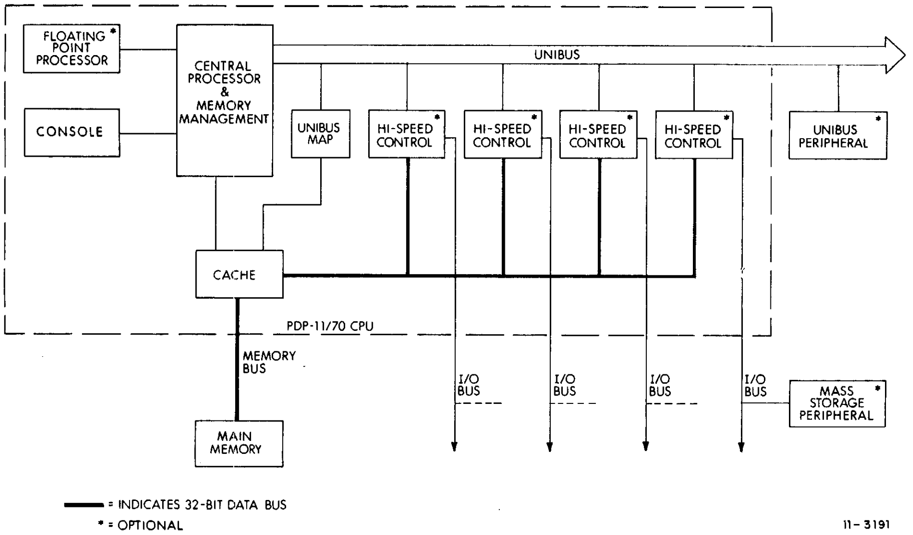


Figure 1-1 PDP-11/70 Block Diagram

The KB11-C Processor performs all arithmetic and logical operations required in the system. Memory Management is standard with the basic computer, allowing expanded memory addressing, relocation, and protection. Also standard is the Unibus Map, which translates 18-bit Unibus addresses to 22-bit physical memory addresses. The Cache contains 2048 bytes of bipolar memory that buffer the data from Main (core) Memory. Main Memory is on its own high data rate bus. The processor has a direct connection to the Cache/Main Memory system for high-speed access.

The PDP-11/70 Console allows direct control of the computer system. It contains a power switch for the CPU. This switch may also be used as the master switch for the system. The Console is used for starting, stopping, resetting, and debugging. Lights and switches provide the facilities for monitoring operations, system control, and maintenance. Debugging and detailed tracing of operations can be accomplished by having the computer execute single instructions or single bus cycles. Contents of all locations can be examined, and data can be entered manually from the Console switches. Console operation and logic are described in Section III of this manual.

Also within the CPU assembly are pre-wired areas for an optional Floating Point Processor, and for up to four optional high-speed I/O controllers (RH70 Massbus Controllers). These controllers have direct connections through the Cache to Main Memory (using the Cache only for timing purposes).

The Unibus remains the primary control path in the 11/70 system. It is conceptually identical with previous PDP-11 systems; the memory in the system still appears to be on the Unibus to all Unibus devices. Control and status information to and from the high speed I/O controllers is transferred over the Unibus. This expanded internal implementation of the PDP-11 architecture has no effect on programming the PDP-11/70.

Three Unibus devices are standard on the PDP-11/70:

1. a KW11-L Line Time Clock

2. a DL11 Synchronous Serial Interface (an LA36 DECwriter II is also standard in the PDP-11/70)

3. a Unibus Terminator and Bootstrap Module.

Also standard are 128KB of parity core memory. Memory, in the PDP-11/70, is not on the Unibus, but on its own high-speed bus (refer to Paragraph 1.2).

1.1.1 Processor

The Processor is the instruction execution section of the system. It implements the PDP-11/45 instruction set. It also acts as the arbitration unit for Unibus control by regulating bus requests and transferring control of the bus to the requesting device with the highest priority.

The Processor contains arithmetic and control logic for a wide range of operations. These include high-speed, fixed-point arithmetic with hardware multiply and divide, extensive test and branch operations, and other control operations.

The Processor is described in Section II of this manual.

1.1.2 Memory Management

Memory Management provides the hardware facilities necessary for address relocation and protection. It is designed to be a memory management facility for accessing all of physical memory and for multi-user, multi-programming systems where memory protection and relocation facilities are necessary.

In order to most effectively utilize the power and efficiency of the PDP-11/70 in medium and large scale systems, it is necessary to run several programs simultaneously. In such multi-programming environments, several user programs could be resident in memory at any given time. The task of the supervisory program would be to control the execution of the various user programs, to manage the allocation of memory and peripheral device resources, and to safeguard the integrity of the system as a whole by control of each user program.

In a multi-programming system, Memory Management provides the means for assigning memory pages to a user program and preventing that user from making any unauthorized access to these pages. Thus, a user can effectively be prevented from accidental or willful destruction of any other user program or of the system executive program.

The basic characteristics of Memory Management are:

16 User mode memory pages

16 Supervisor mode memory pages

16 Kernel mode memory pages

8 pages in each mode for instructions

8 pages in each mode for data

Page lengths from 32 to 4096 words

Each page provided with full protection and relocation

Transparent operation

6 modes of memory access control

Memory access to 2 million words (4 million bytes)

Memory Management is described in Section IV of this manual.

1.1.3 Unibus Map

The Unibus Map is the interface to the Memory System (Cache and Main Memory) from the Unibus. It performs the address conversion that allows devices on the Unibus to communicate with physical memory by means of Non-Processor Requests (NPRs). Unibus addresses of 18 bits are converted to 22-bit physical addresses using relocation hardware. This relocation is enabled (or disabled) under program control.

The top 4K word addresses of the 128K Unibus addresses are reserved for CPU and I/O device registers and is called the Peripherals Page. The lower 124K addresses are used by the Unibus Map to reference physical memory.

The Unibus Map is described in Section V of this manual.

1.1.4 Cache

The Cache is a high-speed memory that buffers words between the processor and Main Memory. The Cache is completely transparent to all programs; programs are treated as if there were one continuous bank of memory.

Whenever a request is made from the Processor to fetch data from memory, the Cache does an address compare to see if that data is already in the Cache. If it is, it is fetched from there and no Main Memory read is required. If the data is not already in Cache memory, 4 bytes are fetched from Main Memory and stored in the Cache, with the requested word or byte being passed directly to the processor.

When a request is made from the Processor to write data into memory:

1. If it is stored in the Cache, it is written both to the Cache and to Main Memory, thus assuring that Main Memory is always updated immediately.

2. If it is not stored in the Cache, it is written only to Main Memory.

Unibus Map references to memory are executed in the same manner as processor references.

Because it stores 1024 words, and because programs tend to use localized sections of code and data, the Cache already contains the next needed word a very high percentage of the time, independently of the program.

The Cache is also the interface between the high-speed I/O controllers and Main Memory.

A detailed description of the Cache is contained in Section VI of this manual.

1.1.5 Unibus

Most of the computer system components and peripherals connect to and communicate with each other on a bus known as the Unibus. Addresses, data, and control information are sent along the 56 lines of the bus. Refer to Figure 1-2.

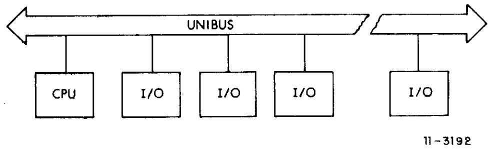


Figure 1-2 PDP-11/70 System Simplified Block Diagram

The form of communication is the same for every device on the Unibus. Peripheral devices use the same set of signals when communicating with the processor, memory, or other peripheral devices. Each device, including memory locations, processor registers, and peripheral device registers, is assigned an address. Peripheral device registers may be manipulated as flexibly as memory by the central processor. All instructions that can be applied to data in core memory can be applied equally well to data in peripheral device registers.

Processor Unibus operations are described in Section II, Chapters 5 and 6 of this manual. Cache Unibus operations are transacted through the Unibus Map (Section V).

1.1.6 Optional Equipment

Floating Point Processor

The Floating Point Processor fits into prewired slots in the Central Processor backplane. It provides a supplemental instruction set for performing single- and double-precision floating point arithmetic operations and floating-integer conversion in parallel with the CPU. The Floating Point Processor provides both speed and accuracy in arithmetic computations. It provides 7 decimal digit accuracy in single word calculations and 17 decimal digit accuracy in double calculations.

Floating point calculations take place in the FPP's six 64-bit accumulators. The 46 floating point instructions include hardware conversion from single- or double-precision floating point to single- or double-precision integers. Refer to the *FP11-C Floating Point Processor Manual* for a detailed description.

High-Speed Mass Storage

Up to four high-speed I/O controllers can be plugged into the KB11-C backplane. A dedicated interface (wired on the backplane) connects these controllers to the memory. A separate bus (Massbus) connects the controllers to high-speed devices. Present DIGITAL devices that utilize this bus structure are the RP04, RS04, RS03, and TU16. The RP04 is a moving head disk pack drive with capacity for 88 million bytes and a transfer rate of 1.25 microseconds per byte. The RS04 is a fixed head disk with a capacity of 1024K bytes and a transfer rate of 1 microsecond per byte (1.2 microseconds at 50 Hz). The RS03 is a fixed head disk, 512K bytes, 2 microseconds per byte. The TU16 is an industry standard 1600 bpi tape unit.

Refer to the following manuals for detailed descriptions of these high-speed devices:

RWS04/RWS03 Fixed Head Disk Subsystem Maintenance Manual

RWP04 Moving Head Disk Subsystem Maintenance Manual

TWU16 Magnetic Tape Subsystem Maintenance Manual

1.2 MEMORY SYSTEM

1.2.1 Representation and Storage

The PDP-11/70 is a 16-bit machine. The data is stored in Main Memory in blocks, each of which consists of two 16-bit words. Thus, the PDP-11 instruction set and the addressing modes are identical to other PDP-11s, but data storage is implemented in a 32-bit configuration. This is transparent to the program and to the processor logic.

The PDP-11 data word consists of two 8-bit bytes, as shown in Figure 1-3. The program addresses either a single byte, when it uses a byte instruction, or a 16-bit word, when it uses a word instruction.

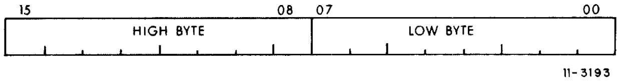


Figure 1-3 High and Low Byte

From the point of view of the program, memory can be viewed as a series of locations, with a number (address) assigned to each location. Thus, a 131,072-byte PDP-11 memory could be represented as in Figure 1-4.

Because PDP-11 memories are designed to accommodate both 16-bit words and 8-bit bytes, the total number of addresses does not correspond to the number of words. A 64K-word memory can contain 128K bytes and consist of 777 777_8 byte locations. Words always start at even-numbered locations.

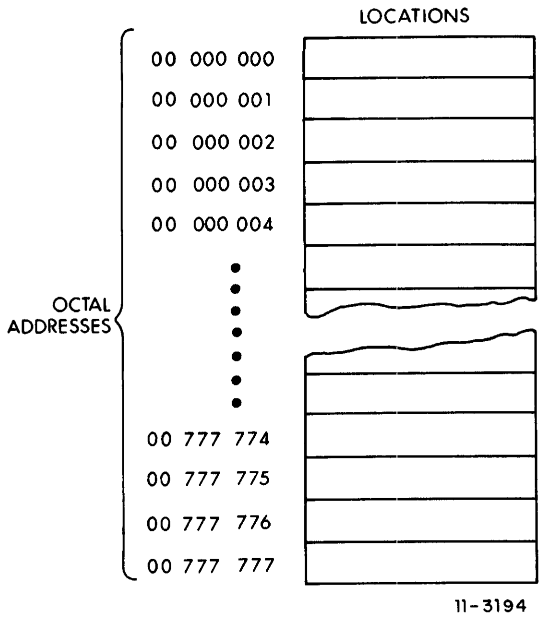


Figure 1-4 Memory Addresses

Low bytes are stored at even-numbered memory locations and high bytes at odd-numbered memory locations. Thus it is convenient, from the point of view of the program, to represent the PDP-11 memory as shown in Figure 1-5.

Main Memory stores data in blocks. A block consists of two 16-bit words (plus 4 parity bits). Figure 1-6 shows how the data for the same memory shown in Figure 1-5 is stored in Main Memory. Block boundaries are located on program addresses whose low-order octal digit is either 0 or 4.

Main Memory addresses are block addresses. The processor and the Unibus use word addresses and the Cache translates these addresses to block addresses.

The Cache, which is the interface to Main Memory for the processor, the Unibus and the high-speed I/O controllers, reads and writes Main Memory as listed below for each of these units:

High-Speed I/O Controllers

1. Read: double word only

2. Write: double word, single word, or byte.

The controllers listed in Paragraph 1.1.6 do not implement byte writes.

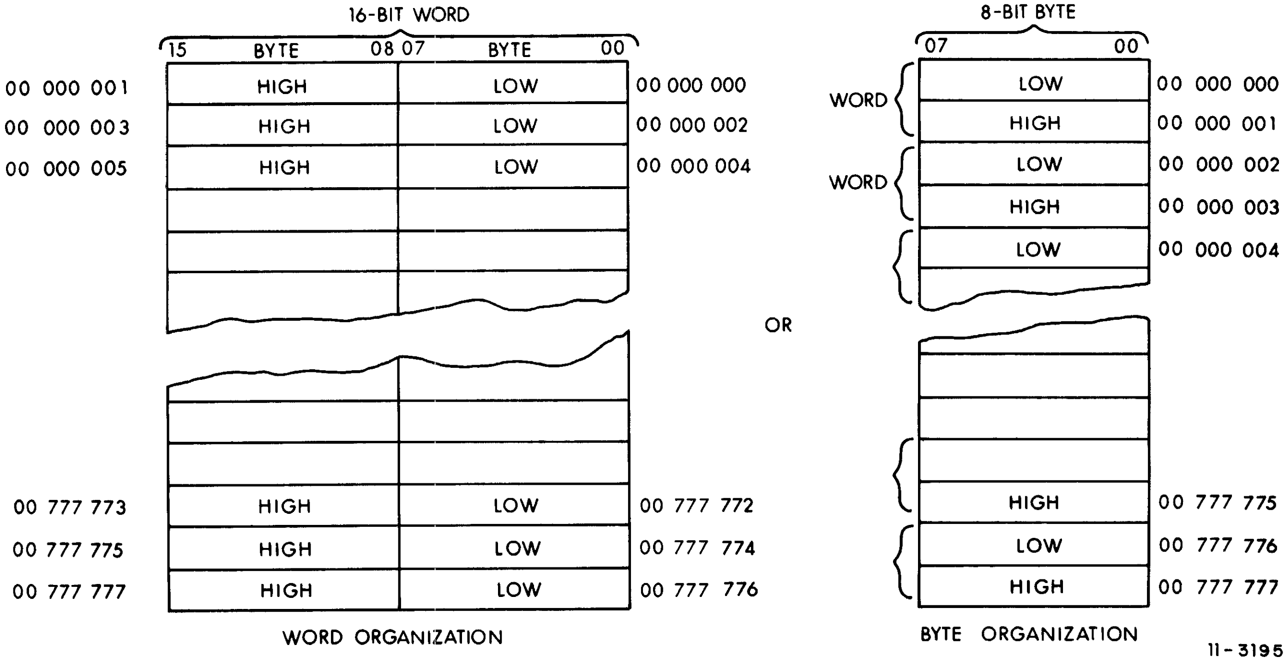
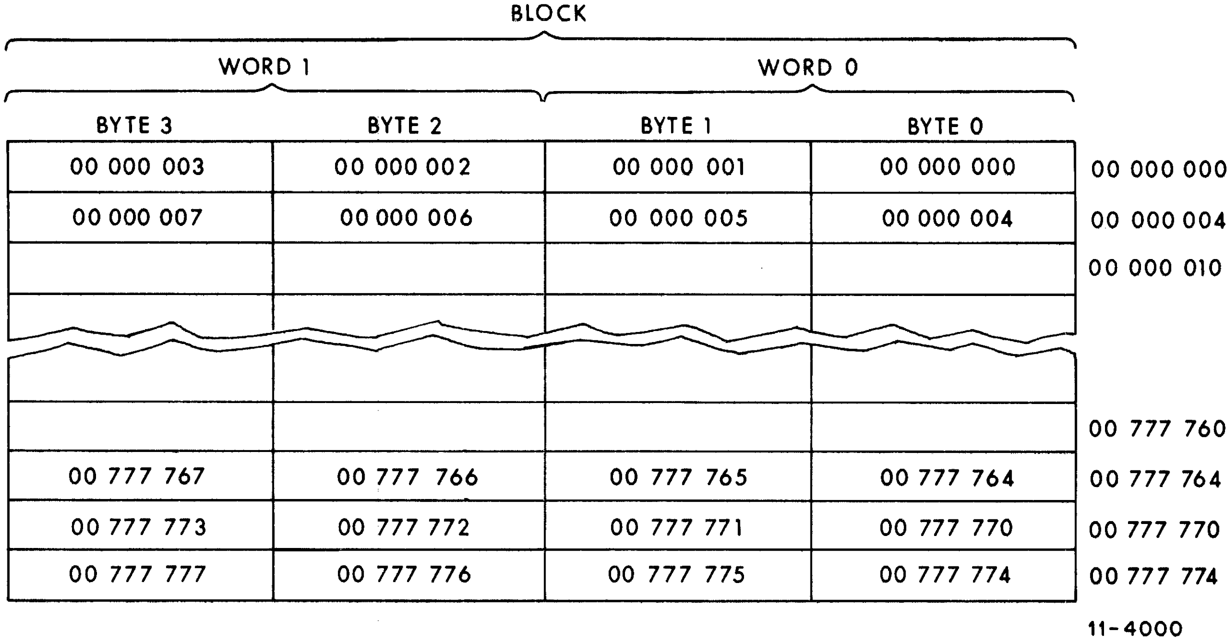


Figure 1-5 Word and Byte Addresses



BLOCK				
WORD 1		WORD 0		
BYTE 3	BYTE 2	BYTE 1	BYTE 0	
00 000 003	00 000 002	00 000 001	00 000 000	00 000 000
00 000 007	00 000 006	00 000 005	00 000 004	00 000 004
				00 000 010
				00 777 760
00 777 767	00 777 766	00 777 765	00 777 764	00 777 764
00 777 773	00 777 772	00 777 771	00 777 770	00 777 770
00 777 777	00 777 776	00 777 775	00 777 774	00 777 774

11-4000

Figure 1-6 Main Memory Addresses

Processor or Unibus

1. Read: double word, but only Word 0 or Word 1 are transmitted to processor or Unibus

2. Write: single word (Word 0 or Word 1) or single byte (one of bytes 0, 1, 2, or 3).

1.2.2 Address Space

The PDP-11/70 uses 22 bits for addressing physical memory. This represents a total of 2^{22} (over 4 million) byte locations.

Three separate address spaces are used with the PDP-11/70. Main memory uses 22 bits, the Unibus uses an 18-bit address, and the computer program uses a 16-bit virtual address. This information is summarized below:

16 bits	program virtual space	2^{16} = 64K bytes
18 bits	Unibus space	2^{18} = 256K bytes
22 bits	physical memory space	4 million bytes

Refer to Figure 1-7. Memory Management generates the physical address output for the processor. This address is an 18-bit address in the case of a Unibus reference and a 22-bit address in the case of a memory reference. The Unibus Map converts 18-bit Unibus addresses to 22-bit Cache addresses.

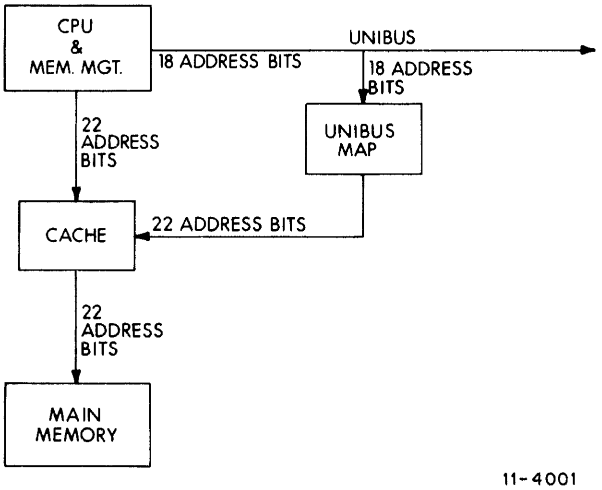


Figure 1-7 Address Paths

Processor Addresses

See Figure 1-8. Of the over 2 million 16-bit word locations possible with the 22-bit physical address, the top 128K are used to reference the Unibus rather than physical memory. Maximum physical memory is therefore $2^{22} - 2^{18}$ bytes, or a total of 1,966,080 words. The system size boundary is the highest address available with the amount of memory included in the system. If the CPU address is between 00 000 000 and the system size boundary, an attempt is made to reference physical memory. Memory addresses between the system size boundary and 16 777 777 are known as Non-Existent Memory (NEXM); any attempt to access these locations is aborted. If the address is in the top 128K, 17 000 000 - 17 777 777, the lower 18 bits of the address are placed on the Unibus.

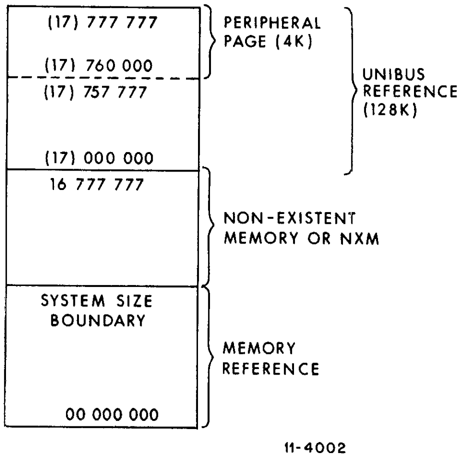


Figure 1-8 Physical Address Space

1.2.3 Mapping

Mapping is the process of converting the virtual address generated by the program to a physical memory address or to a Unibus address, or the process of converting a Unibus address to a physical memory address.

The virtual address is mapped by Memory Management; the Unibus address is mapped by the Unibus Map. Neither of these increases memory access time.

Memory Management and the Unibus Map are separate units and one may be enabled independently of each other. They are both part of the KB11-C and are included in all PDP-11/70 systems.

Refer to Figures 1-9 through 1-11.

1. Mapping of processor addresses is performed in one of three possible ways by Memory Management:

   16-BIT MAPPING
   There is fixed mapping from virtual to physical addresses. The lowest 28K virtual addresses are treated as corresponding to the same physical addresses. The top 4K addresses cause Unibus cycles to addresses 17 760 000 - 17 777 777. Refer to Figure 1-9. 16-bit mapping is enabled after Power Up, Console Start, or the RESET instruction.

   18-BIT MAPPING
   32K virtual addresses for each of the three modes (Kernel, Supervisor, User) are mapped into 128K of physical address space. The lowest 124K addresses reference physical memory. The top 4K addresses cause Unibus cycles to addresses 17 760 000 - 17 777 777. Refer to Figure 1-10.

   22-BIT MAPPING
   This mode produces 22-bit addresses for accessing all of physical memory. The top 128K addresses cause Unibus cycles to addresses 17 000 000 - 17 777 777. Refer to Figure 1-11.

2. Mapping of Unibus addresses is performed by the Unibus Map.

   UNIBUS MAP NOT ENABLED
   When the Unibus Map is not enabled, Unibus addresses 000 000 - 757 777 access memory locations 00 000 000 - 00 757 777, i.e., they are not modified except for the insertion of leading zeroes.

   UNIBUS MAP ENABLED
   When the Unibus Map is enabled, Unibus addresses 000 000 - 757 777 are relocated and a Unibus device may access any location in physical memory.

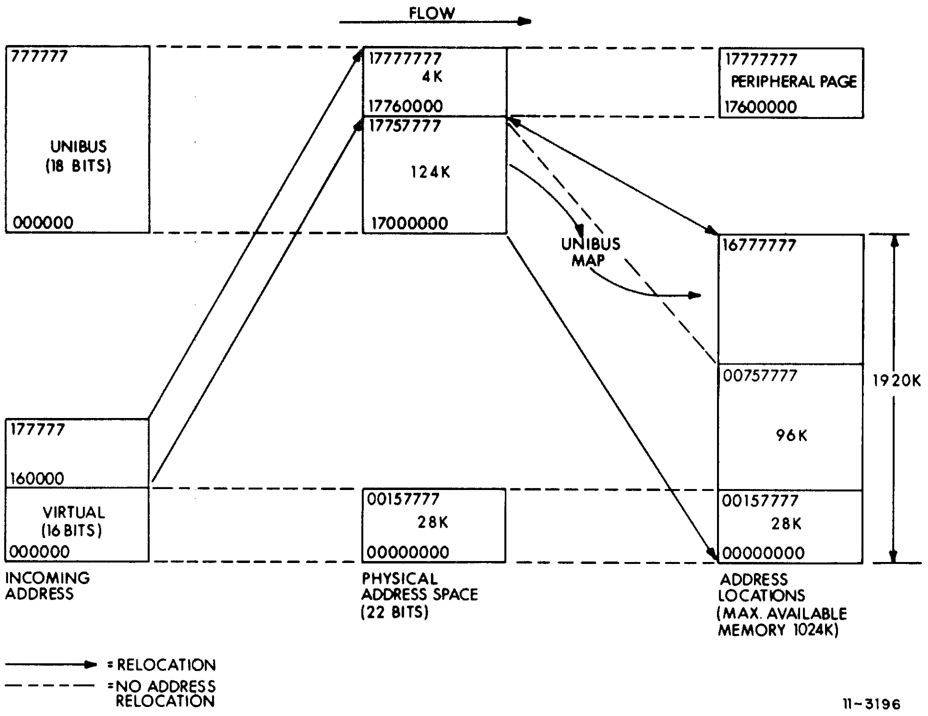


Figure 1-9 16-Bit Mapping

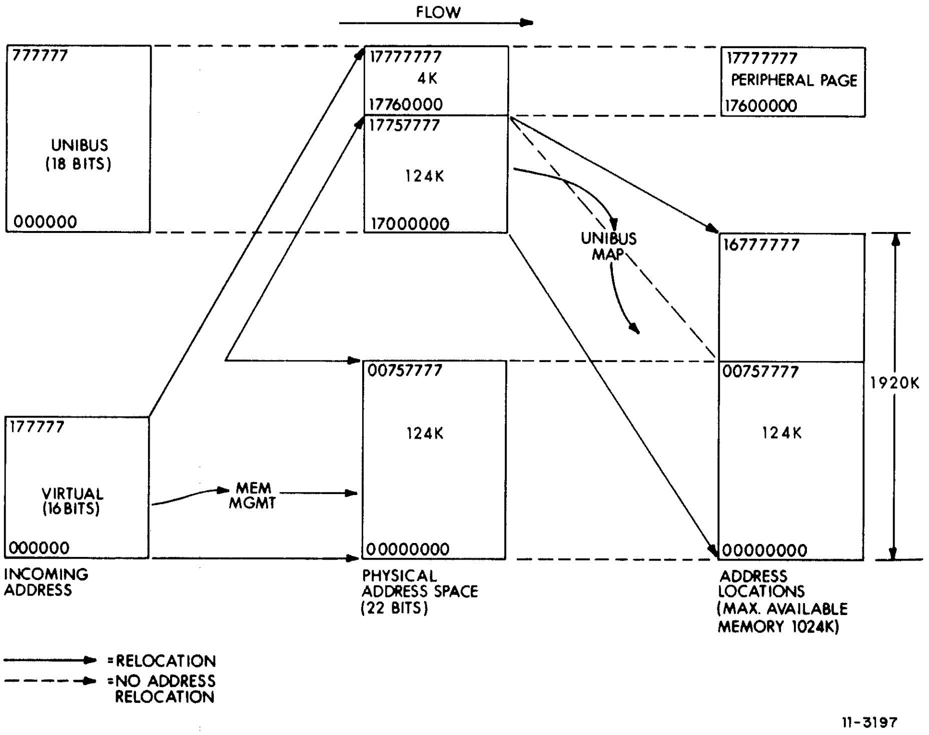


Figure 1-10 18-Bit Mapping

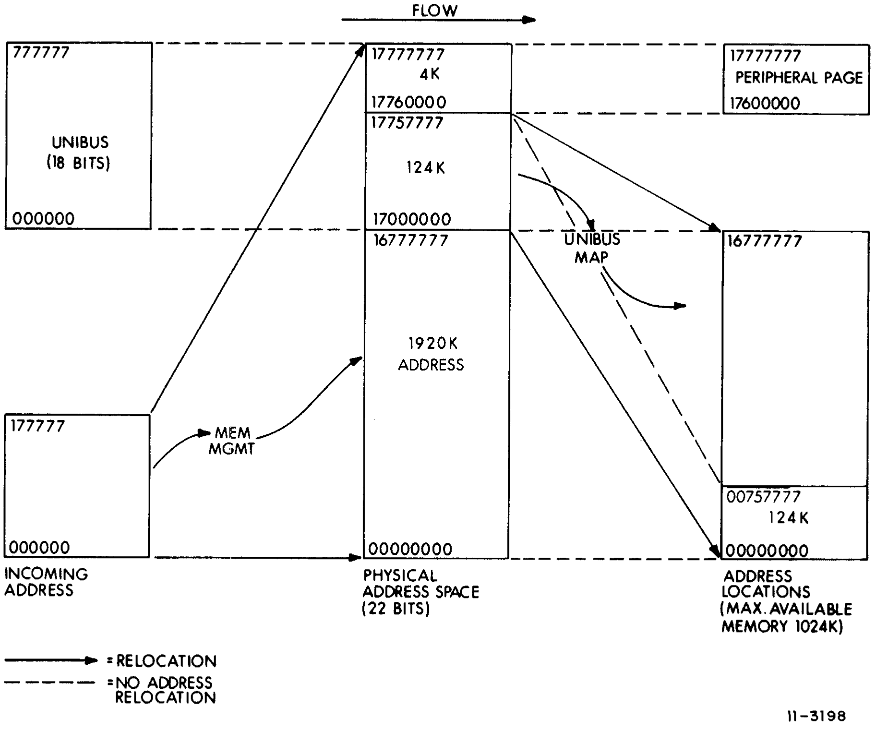


Figure 1-11 22-Bit Mapping

1.2.4 Parity

This paragraph provides general information on parity checking in the PDP-11/70 system. A detailed description of this subject is provided in Section VI of this manual (Cache) and in the Memory Manual.

System Reliability

Parity is used extensively in the PDP-11/70 to ensure the integrity of the data and thus to enhance the reliability of the system. All memory (Cache and Main Memory) has byte parity. Parity is generated and checked on all transfers between Main Memory and Cache, and between Cache and the CPU. It is checked between the high-speed mass storage devices and their controllers, and again between the controllers and core memory. A software routine can be used to log the occurrence of parity errors, to handle recovery from errors, and to provide information on system reliability and performance.

Parity in the System

Main Memory stores one parity bit for each 8-bit byte, (refer to Figure 1-12). The Cache also stores byte parity for data, and in addition it stores two parity bits for the address information (tag storage) associated with each two-word block of data.

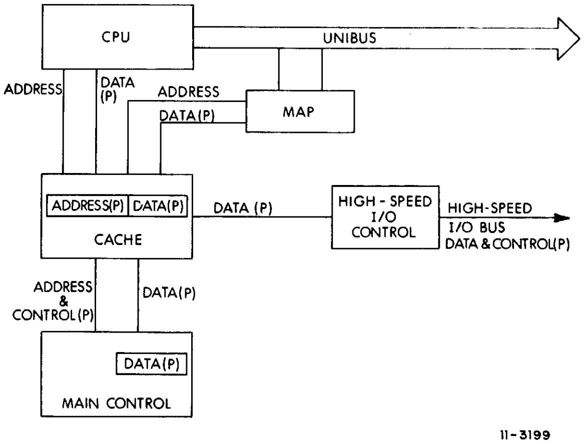


Figure 1-12 Parity (P) in the PDP-11/70 System

The bus between Main Memory and the Cache contains parity on the data lines and on the address and control lines. The high-speed I/O controllers check and generate parity for data transfers to Main Memory, and they have the capability of handling address errors that are flagged by the control in the Cache memory. Refer to Section VI, Chapter 3 for a detailed description of the PDP-11/70 parity system.

System Handling of Parity Errors

The design of the PDP-11/70 allows recovery from parity errors. It also allows operation in a degraded mode if a section of the memory system is not operating properly. This type of operation is possible under program control by using the control registers.

If part or all of the Cache memory is malfunctioning, it is possible to bypass half or all of the Cache. Misses can be forced within the Cache, such that all read data is brought from Main Memory. Operation will be slower, but the system will yield correct results. If part of Main Memory is not working, Memory Management can be used to map around it. If data found in the Cache does not have correct parity, the memory system automatically tries the copy in Main Memory, to allow program execution to proceed. The correct data from Main Memory automatically replaces the data in the Cache which caused the parity error. Therefore, if the error was caused by transitory conditions, it will not occur again.

Aborts and Traps

One of two actions can take place after detection of a parity error: (1) The cycle can be aborted. The computer then transfers control through the vector at location 114 to an error handling routine. (2) The instruction is completed, but then the computer traps (also through location 114). In the first case, it was not possible to complete the cycle; in the second case it was. This second type of parity error usually (but not always) causes the trap before the next instruction is fetched.

CHAPTER 2
CONCEPTS

This chapter introduces several concepts that are useful for the understanding of the KB11-C Processor and the PDP-11/70 system. The first two of these concepts, Microprogramming (2.1) and Parallel Operation or Pipelining (2.2), should be well understood before reading any further. The other two paragraphs, Virtual Machines (2.3) and Reentrant and Recursive Programming (2.4), discuss system concepts that may be easier to understand after a working knowledge of the PDP-11/70 has been acquired. The block diagrams in Appendix A show the interconnection between the several parts of the PDP-11/70, including the RH70 controllers.

2.1 MICROPROGRAMMING

The KB11-C Processor uses a microprogram control section which reduces the amount of combinational logic in the processor. This paragraph introduces the concept of microprogramming by first describing a digital computer, then dividing the computer into various parts, and finally, describing how some of these parts differ for a microprogrammed processor.

Digital Computer Description

Although a computer can effect complicated changes to the data it receives, it must do so by combining a large number of simple changes in different ways. The part of the digital computer that actually operates on the data is the processor. A processor is made up of logical elements; some of these elements can store data, others can do such simple operations as complementing a data operand, combining two operands by addition or by ANDing, or reading a data operand from some other part of the computer. These simple operations can be combined into functional groups; such a group is called an instruction, and it includes operations that read data, operations that combine, change, or simply move the data, and operations that dispose of the data. Instructions can be further combined into programs, which use the combined instructions to construct even more complex operations.

The logical elements of a processor can only perform a small number of operations at one time. Therefore, to combine operations into an instruction, the instruction is divided into a series of operations (or groups of operations that can be performed simultaneously). The processor does each part of the series in order. One way to describe how the processor executes an instruction is to call each operation (or group of operations) a machine state. An instruction then becomes a sequence of machine states which the processor enters in a specific order.

The processor can be completely described in terms of machine states by listing all the machine states in which the processor can perform (i.e., all the different operations or groups of operations that it can perform) and all the sequences in which these machine states can occur. The sequence of machine states is determined by the current state of the computer; this includes such information as the instruction being executed, the values of the data being operated on, and the results of previous instructions.

In terms of the machine state description, the processor can be divided into two parts. The first part, called the data section, includes the logic elements that perform the operations which make up a machine state. The second part, called the control section, includes all the logic that determines which operations are to be performed and what the next machine state should be. The data section and control section are discussed in the following paragraphs.

The data section in the KB11-C is usually referred to as the *Data Paths* and is described in Section II, Chapter 2. The control section is described in Section II, Chapter 1, Instruction Decode and Microprogram Control.

The Data Section

During each machine state, the data section performs operations selected by signals from the control section. The data section provides inputs to the control section which help to determine the next machine state; the data section also exchanges data with other devices external to the processor.

The data section can be divided into three functional sections; each section is discussed in one of the following paragraphs.

The Data Storage Section

For the processor to combine data operands it must be able to store data internally, while simultaneously reading additional data. Often, a processor stores information about the instruction being executed, about the program from which the instruction was taken, and about the location of the data being operated on, as well as a number of data operands. When the processor must select some of the internally-stored data, or store new data, the control section provides control signals which cause the appropriate action within the data storage section.

The Data Manipulation Section

This section includes the various logic elements that actually change data. Many of these elements are controlled by signals from the control section, which select the particular operation to be performed. Data manipulation is performed on data being transferred between the processor and the rest of the system, and on data that remains within the processor. In some cases, the data that remains within the processor is used to control the processor by providing inputs to the sensing section of the processor control.

The Data Routing Section

The interconnections between the logic elements in the data storage section and the elements in the data manipulation section are not fixed; they are set up as required in each machine state. The control section generates signals that cause the logic elements in the data routing section to form the appropriate interconnections within the processor, and between the data interface and the data storage and manipulation sections.

The Control Section

The control section of a processor receives from the data section, inputs which are used by the sensing logic to help select the next machine state. The control section also generates control signals to all parts of the data section and communicates with other parts of the computer system through control signals. The following paragraphs describe the three parts of the control section.

The Sequence Control Section

The primary control of the processor is the selection of the sequence of machine states to be performed. This is done by the sequence control section which selects the next machine state on the basis of:

1. the current machine state

2. inputs from the data section (such as the instruction type or the data values)

3. information about external events.

The sequence control section maintains information about the current machine state, and receives information from the data section and the external environment through the sensing section.

The Function Generator

In each machine state, the data section performs operations selected by signals from the control section of the processor. The function generator produces these control signals on the basis of the current machine state and also on the basis of inputs from the sensing section, such as information on the instruction type.

The Sensing Logic

In general, the sequence control section requires in ·puts that select one of a limited number of machine states to follow the current state.

The Control Section in the KB11-C

The function generator comprises the microprogram Read Only Memory (ROM), its output buffer, and several logic elements that generate control signals based on sensed inputs (notably through the subsidiary ROMs). The sequence control comprises the microprogram address generation logic. The sensing section includes the various logical elements that receive inputs from the data section, especially the condition-code generator, the subsidiary ROMs, and the branch logic.

Microprogramming in the Control Section Implementation

This paragraph describes two methods of implementing the control section of a processor. The first method, which is called the *conventional* method for the purposes of this discussion, uses combinational networks, with many inputs combined in varying ways to produce each output. The second method, which is called *microprogramming,* replaces most of the combinational networks with an array structure. The array requires a small number (approximately 10) of inputs to select the output states for a large number (approximately 100) of signals. Because the array is a regular structure, it is simpler to construct and understand, and less expensive.

Conventional Implementation

In a conventional processor, each control signal is the output of a combinational network that detects all the machine states (and other conditions) for which the signal should be asserted. The machine state is represented by the contents of a number of storage elements (such as flip-flops), which are loaded from signals that are, in turn, the outputs of combinational networks. The inputs to these networks include:

1. the current machine state
2. sensed conditions within the Processor
3. sensed external conditions.

The number of logical elements in the processor is often reduced by sharing the outputs of networks which generate intermediate signals needed in the generation of several control signals, or even in the generation of control signals and machine states. Unfortunately, while this reduces the size of the processor, it increases the complexity and difficulty of understanding the device because it is no longer obvious what conditions cause each signal. In additon, the distinction between the sequence control and the function generator is blurred, which makes it more difficult to determine whether improper operation is caused by a bad machine state sequence or, more simply, by the wrong control signals within an otherwise correct machine state.

Microprogrammed Implementation

The microprogrammed implementation is based on the following observation. Each control signal is completely defined if its value is known for every machine state. The function generator section can therefore be implemented as a storage device: the storage is divided into words, with each word containing a bit for every control signal; there is one word for each machine state. During each machine state, the contents of the corresponding word in the storage element are transmitted on the control lines. For most control signals, the output of the storage unit is the control signal; no additional logic is required.

The two tasks of the sequence control section are to select the next machine state, and to provide information about the current machine state to the function generator. The only information that the function generator in a microprogrammed processor requires is which word to use as control signals. Therefore, the seqence control simply provides an address that selects the correct word. The sequence control must also select the address of the next word to determine the machine state sequence. Because the next machine state is determined in part by the current machine state, information is stored in the microprogram that helps to select the next state; the microprogram word contains the control signal values and the address and sensing control information required by the microprogram address generation logic (i.e., by the sequence control).

In a microprogrammed control like the one described above, the two major portions of the control section have been simplified to regular logical structures. The function generator is entirely separate from the sequence control, so it is easy to isolate malfunctions to the microprogram storage or to the address generator. In addition, the sensing logic is simplified, because each sensed condition is reduced to a single signal and the sensing logic selects the appropriate signals for the current machine state, based on signals output from the microprogram storage. To summarize this discussion, a microprogrammed processor has a simpler, more regular, more easily repaired control structure, based on the generation of control signals from stored information, and the selection of each machine state, based on information stored in the current machine state, and on information from a simplified sensing section.

2.2 PARALLEL OPERATION (PIPELINING)

In a digital computer system, the processor is usually the fastest part of the system. In order to achieve the maximum speed of operation, all parts of the processor should be used as much as possible. To prevent the processor from wasting time waiting for other parts of the system, the processor must make use of the external data transfer interface as much as possible. Because any one operation that the processor performs uses only part of the processor's available resources, the two considerations above require the processor to perform several operations in parallel.

In general, the sequence of operations required for each instruction uses various parts of the processor at different times. Some parts of the processor, such as the program counter, are used only during the early parts of the instruction; others, like the shift counter, are used only during later parts of the instruction. The processor can be fully utilized only if different parts of the processor can be used for parts of different instructions during the same machine state.

When the processor works on the early part of an instruction at the same time that it completes the previous instruction, this form of parallel operation is called *pipelining*. The processor attempts to make continuous use of the external data interface by fetching each word addressed by the Program Counter (PC) in succession (incrementing the PC during each transfer), on the assumption that the next word required will be the one following the current instruction. In the pipelining analogy, the processor attempts to fill a pipe, corresponding to the different parts of the processor used successively by each instruction, with a series of instructions.

The current instruction often requires some other words from the external storage. At times, the next instruction does not follow the current instruction because the PC has been explicitly changed by the current instruction. When either of these two conditions occurs, the processor must stop the data transfer begun after the instruction fetch and begin a data transfer with a different address. In the pipeline analogy, this is a break in the smooth flow of instructions through the pipe; some time is lost before the pipe drains (the current instruction is completed) and can be refilled (a new instruction fetched and a transfer begun to read the word following that instruction).

A second form of parallel operation occurs in the KB11-C to further improve the utilization of the processor. Because the processor includes several types of data storage and data manipulation elements, with different interconnections, several data transfers can take place within the processor simultaneously. As an example, during the same machine state that completes an external data transfer, the processor can read a general register into a temporary storage register, and perform an addition that adds a constant to the program counter.

The use of parallel operations within an instruction reduces the number of machine states (and therefore the total time) required to execute each instruction; the use of pipelining further reduces the number of machine states required to execute a program by effectively eliminating the elapsed time between many external data transfers.

2.3 VIRTUAL MACHINES

The processor executes instructions and operates on data, both of which are stored in memory, and it responds to various asynchronous events.

The response to an interrupt or trap is not entirely designed into the processor. Instead, the response is controlled by a series of instructions (a program) which is selected by a simpler hardware response when the asynchronous event is detected. Often, a number of programs are required to respond to a number of events, and the scheduling, coordination, and interaction of these programs is one of the most important (and difficult) parts of programming a computer system.

In many applications, the user programs that are written for the system are treated as though they are interrupt response programs. This is done to simplify the scheduling, to allow each user program to operate with a terminal (some form of character I/O device), and to allow several user programs to operate at once. By running several programs at once, the processor can be utilized more fully than is generally possible with only one user program, which would often be waiting while devices other than the processor completed data transfer operations. With several programs to be run, the processor can be switched among the programs so that those ready to run have the use of the processor while others are waiting. The use of the processor for several programs at the same time is called *multiprogramming*.

Running programs in a multiprogrammed system presents several difficulties. Each program can be run at arbitrary times, but all the programs must be capable of running together, without conflict. A failure in one program must not be allowed to affect other programs. Each program must be able to use all features of the system in a simple, easily-learned manner, preferably in such a way that the program does not need to be modified to run in a different hardware configuration.

These difficulties are overcome by providing each program with a virtual machine. The programmer writes his program as though it is to run by itself; the program uses any system resources (such as memory or peripheral devices), and the system provides the services necessary to support the program and coordinate it with other programs in operation. The physical hardware in the system is combined with a control, or executive program, to simulate a more powerful hardware machine; it is for this more powerful, but abstract, machine that the programs are written.

Based on this discussion, the hardware machine and the executive program must combine to fulfill the following four major objectives of the virtual machine:

1. Mapping – The virtual machine of the program currently in operation must be assigned to some part of the hardware machine.

2. Resource management – The scheduling of programs, and the allocation of parts of the hardware machine, must be performed by the executive program.

3. Communication – The virtual machine must be able to request services from the executive program, and the executive program must be able to transfer data back and forth with the user programs.

4. Protection – The system that supports the virtual machine, and all other virtual machines, must be protected from failures in any one virtual machine.

Each of these subjects is discussed in one of the following paragraphs.

Mapping

Each time a program is run (or, if the multiprogramming system is running several programs in a round-robin manner, each time a program resumes operation), it has some of the system dardware allocated to it. This generally includes some part of the memory to contain the instructions and data required by the program, some of the processor's registers, a hardware stack (which is actually an area in the memory and a pointer to that area in a processor register), possibly some peripheral devices, and perhaps a fixed amount of the processor's time. All of thse allocations must be made in such a way that the hardware machine can then execute the user program with a minimum of extra operations; i.e., so that the execution of the user program requires as few additional memory cycles, or additional machine cycles, as possible. Therefore, the allocation is done entirely in the hardware machine; registers in the hardware contain all the allocation (mapping) information, and all references to virtual addresses, virtual stack locations, virtual register contents, or virtual devices converted by hardware to physical references.

In a PDP-11/70 System, mapping is done by two devices. The mapping of virtual registers into processor registers, of the virtual stack, and of the virtual program counter, is done by loading the appropriate values into the processor registers; one of two sets of general registers can be selected for the user, and the processor has a separate stack pointer for user mode, while the program counter is changed by interrupt and trap operations and by the Return from Interrupt (RTI) or Return from Trap (RTT) instructions.

The remaining mapping functions distribute the virtual memory into the physical memory. In the physical memory, many specific addresses are reserved for special functions; the lowest addresses are used for interrupt and trap vectors, while the highest addresses are used for device registers. Because all functions that require reserved addresses in the physical memory are performed either by the physical machine or by the control program, these addresses need not be reserved in the virtual machine. Therefore, the programs written to be run in the virtual machine can use any addresses; specifically, these programs can start at address 000000 and continue through ascending addresses to the highest address needed.

In discussions of the virtual memory and the physical memory, it is often necessary to describe the addresses used to select data items within the memory. The range of addresses that it is possible to use is called the *address space*. The maximum range of addresses that can be used in the virtual machine (which in the PDP-11/70 is the maximum number that can be contained in a 16-bit word) is called the virtual address space, while the maximum range of physical addresses that can exist in the hardware system is called the physical address space (in the PDP-11/70 this can be all the addresses expressed by a 22-bit number).

If the user program is to use addresses in the virtual address space that are reserved in the physical address space, the virtual address space must be relocated to some other part of the physical address space. In a multiprogramming system, several user programs, each in its own virtual address space, may be sharing the physical address space. Therefore, the relocation of the virtual address space into the physical address space must be variable; each time a program is run, it may be allocated a different part of the physical address space. Memory Management provides the capability of varying the relocation for each user program by storing a map of the memory allocation in a set of registers.

Resource Management
In a multiprogramming system, each user program operates in a virtual machine that can utilize any of the possible devices or functions of the physical machine, as well as many functions performed by the executive program. The resources that exist in the system must be allocated to each user program as required, but without allowing conflicts to arise where several user programs require the same resources. The physical machine and the executive program must resolve any protective conflicts by scheduling the resources for use by different programs at different times, and must schedule the user programs to operate when the resources are available.

The management of input/output or peripheral devices is beyond the scope of this discussion, which is primarily concerned with the basic PDP-11/70 System. Within the system, the two most important resources which require the most care and effort to control are the memory and the processor.

Processor Management
The processor can only execute one instruction at a time. When several programs are sharing the use of the processor, the processor operates on each program in turn; either the processor is shared among the programs, by using periodic interrupts to allow the executive program to transfer the processor to another user program, or each user program runs to completion before the next user program begins. To share the processor on a time basis, the executive program must perform the transfer from one virtual machine to another. Each virtual machine is given control of the physical machine by loading the map of that virtual machine into the physical machine. That is, the executive program changes virtual machines by changing the contents of the processor registers used by the virtual machine, and by changing the contents of the registers in Memory Management which map the virtual address space.

Memory Management
The following discussion assumes that Memory Management is enabled. Memory Management is much more complicated than Processor Management. If a program uses a large proportion of the virtual address space, and only a small amount of memory is physically available in the system, the program may be too large to fit into the memory all at once. Fortunately, in most programs only a small part of the program (or possibly several small parts, one for the instruction stream and one or more for blocks of data) is used at any one time. To take advantage of this fact, the virtual address space is divided into *pages* so that each page can be mapped separately. Only the pages that are in use in the current instruction are required to be in the physical memory during the execution of that instruction.

A system which uses Memory Management to permit each virtual machine to have a larger address space than the available physical memory must also include a mass storage device to hold those parts of each virtual memory that are not in the physical memory. As a program proceeds through a sequence of instructions, it requires different pages of the virtual memory. The memory map in the Memory Management includes relocation information for each page of the virtual address space, and also includes information specifying which pages are currently in the physical memory. If the processor attempts to perform transfers with a virtual address which is on a non-resident page, the instruction is aborted. A part of the executive program which transfers the required page into the physical memory and changes the map in Memory Management to reflect the newly available page is then executed.

Memory Use Statistics

If it is necessary for the executive program to bring a page into the physical memory, but all of the physical memory is already in use, the executive program must remove another page (from the same virtual machine or, in a multiprogramming system, from some other virtual machine) from the physical memory. When a page is removed from the physical memory, a copy of that page must be stored in the mass storage device; if a copy of the page is already on the mass storage device, and none of the data (or instructions) stored on the page have been changed, the writing of the page onto the mass storage device can be bypassed. Each time a page must be replaced, the executive program attempts to predict which page is least likely to be used in the future, so that it will not soon need to be moved back into the physical memory.

Memory Management includes hardware to permit choosing the page to be replaced and to determine whether that page must be written onto the mass storage device. Each external data transfer performed by the processor requires that Memory Management convert a virtual address into a physical address and keep track of which virtual pages have been accessed and which virtual pages have been written into. The executive program operates on the assumption that pages which have been recently accessed will also be used soon. To find a page which can be replaced, the executive program looks for a page which has not been used, preferably from the address space of a user other than the current user. If there are no virtual pages currently in the physical memory that have not been accessed, the executive program looks for a page that has not been written into, to avoid having to copy a page to the mass storage device. If all the virtual pages in the physical memory belong to the current user, the executive program looks for a page that has not been used recently, again preferably one that has not been written into. By use of the hardware Memory Management unit and of a variety of scheduling and allocation algorithms in the executive program, the system can provide a number of user programs with virtual machines of great power and flexibility, with a minimum burden on the user program.

Communication

A program running in a virtual machine must be able to communicate with the executive program, to request various services performed by the executive program, or to determine the status of the system. The same type of communication can be used for communication between virtual machines, by providing inter-machine communication as a service through the executive program. The same hardware functions that provide a means for the user program to communicate to the executive program are also used by the executive program to determine the status of the user program when a trap or abort condition occurs.

The user program requests services by executing trap instructions (such as EMT, TRAP, or IOT). Abnormal conditions caused by a program failure, such as an odd address for a word data transfer, or an attempt to execute a reserved instruction, cause internal processor traps. In either case, the trap function performed by the processor serves to notify the executive program that an instruction is required.

Context Switching

The executive program must then begin executing instructions to perform the requested service or to correct the failure condition, if possible. However, in order for the hardware machine to operate on any program other than the user program, the mapping information must be changed to reflect the allocations used by the new program.

The trapping function performs the change of most of the mapping information. The contents of the Program Counter (PC) and the Processor Status (PS) registers are changed directly; the old contents are stored on a stack in memory, pointed to by a stack pointer, and the new contents are supplied from locations called a trap vector. The address of the trap vector is provided by the processor and depends on the type of trap instruction or trap condition, so that for each trap instruction or condition, a different PC and PS can be supplied.

Memory Management stores the maps for the executive program and one user program in separate registers. The processor indicates which map should be used to relocate virtual addresses. During the execution of instructions (as opposed to the interrupt and trap service function), the address space map to use is specified by bits 15 and 14 of the PS. These bits also specify which Stack Pointer (SP) register in the processor to use (there is a separate SP for each virtual machine). Because the trap and interrupt service function loads the PS register with a new value, this function changes almost the entire virtual machine context directly.

The only remaining parts of the virtual machine context that require changes are the general registers in the processor. These can be changed either by saving the contents of the registers from the previous virtual machine on the hardware stack and loading new contents, or by selecting the alternate set of general registers (the processor has two sets of general registers, 0 – 5). Register set selection is controlled by bit 11 of the PS register, so this method can be used in conjunction with the trap service function.

To summarize the change of virtual machines: the mapping in the hardware system includes the selection of a register set, a stack pointer, a program address (in the program counter), an address space, and a processor status. The trap and interrupt service function, which is performed by the processor as an automatic response to trap an instruction or abnormal condition, can change all of these selections as follows:

1. The program counter and processor status are changed directly.

2. Bits 15 and 14 of the new PS select the new address space and stack pointer.

3. Bit 11 of the new PS selects the new register set.

The mapping and selection information for the previous virtual machine is completely saved, either by remaining in unselected portions of the processor and the Memory Management unit, or by being stored on the hardware stack. If the selected register set is shared with other virtual machines, the register contents must be changed by an instruction sequence.

Inter-Program Data Transfers

When the new virtual machine begins executing a service program for the programmed request (if a trap instruction was executed) or abnormal condition (if a trap condition occurred), the service program must get information from the previous virtual machine. This information may define the status of the previous virtual machine, after an abnormal condition occurred, so that the service program can correct the condition and restore the correct status before returning control to the previous virtual machine. If the service program is performing a service, the information required from the calling program may define the specific type of service to perform, or provide the addresses of data buffers, or specify device and file names.

Most information required by the service program is stored in the calling program's address space. To get this information, and to return information to the calling program, the service program must be able to operate in the present address space and transfer data in the previous address space, at the same time. The **KB11-C** Processor provides instructions to do this.

The special instructions that transfer data between virtual address space make use of the PS register to specify which address space is being used by the current virtual machine, and which address space was used by the previous machine (this is identified by bits 13 and 12 of the PS). The data is transferred between the hardware stack of the current address space and arbitrary addresses of the previous address space. The calculations of the virtual address in the previous address space; i.e., any index constants or absolute addresses used to generate the virtual address, are taken from the current address space, just as the instructions are.

Each virtual address space is divided into an Instruction (I) space and a Data (D) space. Each I or D space has a full set of 2^{16} virtual addresses. Therefore, the communication instructions are available in two versions; one to transfer with the previous I space, and one to transfer with the previous D space. A different instruction is needed for each transfer direction as well, so there are four communication instructions: Move To Previous Instruction (MTPI) space, Move To Previous Data (MTPD) space, Move From Previous Instruction (MFPI) space, and Move From Previous Data (MFPD) space.

Returning to the Previous Context

Because all the mapping and context information for the previous virtual machine is saved when the trap and interrupt service function sets up a new virtual machine, the hardware system can resume the execution of any program at the same point that it was interrupted. This is done with a Return from Interrupt (RTI) or Return from Trap (RTT) instruction, which replaces the PC and PS values of the current virtual machine with the stored values from the previous virtual machine.

The PS selects most of the mapping information, as described previously, so the return instructions completely restore the previous context.

Protection

The hardware system and the executive program must be protected from failures in each virtual machine. In addition, most systems provide protection so that no program operating in a virtual machine can take control of the system or affect the operation of the system without authorization. A third form of protection that is useful in a large and complex system is the protection of the executive program against itself. The executive program is divided into a basic, carefully written Kernel, which is allowed to perform any operation, and a broader Supervisor, which cannot perform privileged operations, but which provides various services useful to the executive program and to the user programs.

The forms of protection provided include the different address spaces for different types of programs, a variety of restricted access modes, and restricted processor operations. The address space protection can be used with any type of program, whether operating in User, Kernel, or Supervisor mode. The restricted processor operations are usable only in Kernel mode; Supervisor mode has the same restrictions as User mode.

Separate Address Spaces

The most basic protection against modification of the executive program by a User program (or of the Kernel section by the Supervisor section) is the separation of the address spaces. A program operating in User mode operates in the User address space. It cannot access any physical addresses that are not in that address space, regardless of their correspondence to addresses in any other virtual address space. The executive (Kernel) program can prevent a User program from accessing other virtual address spaces through the communication instructions (MTPI, MTPD, MFPI, MFPD) by forcing bits 13 and 12 of the stored processor status word to 1s (to reflect User mode) before executing an RTI or RTT instruction to return control to the user program. This forces the previous mode bits in the PS register to take on User mode, just as the current mode bits are set to User mode, and the communication instructions operate only within the User address space .

Access Modes

Within one address space, it is often useful to be able to protect certain parts of a program from unintentional modification. This can be done by allowing the data in those addresses to be read, but prohibiting transfers into the addresses. This is known as read-only (or write-protected) access. Areas in a virtual address space that contain alterable data must permit read/write access, but areas that contain unmodified instructions may be read-only.

Another useful form of access protection distinguishes between read accesses that fetch instructions (or address constants) and any accesses that transfer data. If instructions can be accessed by the processor only as instructions, they can be executed but they cannot be read or transferred to any other part of the address space. This prevents the user from determining what the instructions are in order to tamper with the instruction sequence or attempt to modify the program in undesirable ways. This type of access restriction is called execute-only access.

Memory Management provides a read/write, read-only, and execute-only access modes system. The access mode is stored in the mapping registers along with the relocation information; in fact, when a page of the virtual address space is not in memory, a special access code that identifies the page as non-resident is used. The execute-only access mode is not a separate access mode, but is provided by separating the address space into two address spaces that are used for the different kinds of transfers. One address space is used for all transfers that fetch instructions and is called the Instruction (I) space, while a second address space is used for all data transfers and is called the Data (D) space. If the two address spaces are mapped separately, attempts to use the same address for an instruction and for data may address different physical locations. If no addresses in the D space correspond to the physical addresses used in the I space, the instructions cannot be accessed as data and an execute-only access mode has been achieved. This mode must be used with caution: tables that are accessed by indexed address modes must be in D space and MARK instructions, which are stored on .he hardware stack as data and then executed, and require the stack to be in the same virtual addresses in I and D space.

Privileged Instructions

Certain PDP-11 instructions that affect the operation of the hardware machine must be prohibited in the virtual machine. These include the HALT instruction, which stops the physical machine and thus prevents any virtual machine from operation, the RESET instruction, which stops all input/output devices, regardless of which virtual machine they are allocated to, and various PS change instructions. These instructions are allowed only in Kernel mode so that the executive program can control the entire hardware system; they are ineffective in the Supervisor or User mode. The RESET and Set Priority Level (SPL) instructions are allowed to execute in these modes, but have no effect; the HALT instruction activates a trap function so that the executive program may stop all action for the virtual machine that executed the HALT, but not for other virtual machines.

2.4 REENTRANT AND RECURSIVE PROGRAMMING

A program can generally be divided into routines, each of which performs a function that is built up from a sequence of instructions. Often, the function performed by a routine is needed in several other routines, so it is desirable to be able to call the routine from many other routines in the program; i.e., the program should be able to transfer the processor to the instructions following the calling instruction. A routine which is called from other routines is said to be subordinate to those routines and is called a subroutine; the special instructions that transfer the processor to the beginning of a subroutine and that return the processor to the calling routine are called subroutine linkage instructions.

Recursive Functions

Some procedures are most easily implemented as a subroutine that either performs a part of the procedure and then calls itself to perform the rest of the procedure, or completes a computation and returns a partial (and finally, a complete) result. This is called recursive operation. The common example of a recursive procedure is one that calculates the factorial of a number (the factorial is the product resulting from the multiplication of a number, n, by all smaller numbers). The recursive procedure to calculate a factorial of a positive integer is as follows:

1. If n is 1 or 0, return 1 as the value of factorial n.

2. If n is greater than 1, compute the factorial of n minus 1, multiply that number times n, and return that value.

For example, to compute the value of factorial 3, the procedure is to compute the value of factorial 2 and multiply by 3. However, the value of factorial 2 is the value of factorial 1 times 2. The value of factorial 1 is found by Step 1. to be 1, so the final result is 1 times 2, multiplied by 3, or 6. The same recursion computes the factorial of any positive integer, in n recursions for a number n.

Use of a Stack in Recursive Routines

When a subroutine is called recursively, the linkage information for each call (the information required to return to the calling program) must be saved during subsequent calls. Since a recursive subroutine can be called again before it returns from the first call, the linkage information should not be stored in a fixed location; instead, it is stored in a stack, with each linkage in a different location and a pointer that identifies the specific location for each linkage.

Assume that subroutine A calls subroutine B, which then calls subroutine C. Subroutine C must return control to subroutine B before subroutine B can return control to subroutine A. It can be seen that in this case the last linkage which has not been used for a return must be the first one used; i.e., the linkages must be used in a last-in, first-out sequence. A storage area whose locations are used for last-in, first-out storage is called a stack; a pointer is used to point to the last entry placed on the stack, and the subroutine linkage instructions that put information on the stack (a push operation), or remove information from the stack (a pop operation), change the contents of the pointer so that it always points to the correct word for the next linkage operation.

One of the **KB11-C** processor's general registers is used by the subroutine linkage instructions as a stack pointer. This register is the Kernel Stack Pointer (SP) and it must be initialized to point to the first word in a stack area. This same stack is also used for storage of context or linkage information by the trap and interrupt service function, which is described in Section II, Chapter 6. The traps, interrupts, and subroutine calls are all handled in the same last-in, first-out manner.

A subroutine that can be called recursively should not move data into fixed locations, because later executions of the same subroutine (before the current execution is finished) may also execute the same data transfer instructions. The best way to keep the data storage for each execution of a subroutine separate is to store the data on the stack in the same manner as the linkage information.

Reentrant Functions

Keeping the data storage separate from the program is particularly important for programs and subroutines that can be called from more than one virtual machine. If several virtual machines are executing the same program, it can be called from more than one virtual machine. If several virtual machines are executing the same program, it is desirable to have only one copy of the program in the physical memory, and to map each virtual address space into the same physical address space. However, in a muliprogramming system, one virtual machine may begin execution of a program and then be interrupted; a second virtual machine may begin execution of the same virtual program and then run out of time; the original virtual machine may resume execution and complete the program; and the second virtual machine may resume executions. The programmer cannot make any assumptions about where each virtual machine may resume execution, nor can he make any assumptions about where each virtual machine stops, so the program must be capable of being reentered at any time, regardless of what other virtual machines have done with the program.

Programs designed to store all their data on a stack, so that each virtual machine that uses the program simply uses a different stack, are called reentrant programs. A different stack pointer is selected each time a different virtual machine is selected. If the executive program changes the context of the user virtual machine, to run a different user, it changes the address mapping of the stack area and the contents of the stack pointer, so that each activation of a program executes the program in complete isolation from other activations by other virtual machines.

Indexed Addressing of Parameters

When a program or routine calls a subroutine, the calling routine may send data to the subroutine. The amount of the data to be "passed" to the subroutine may vary, as may the amount of data returned by the subroutine. By placing all the data on the stack, the amount of data becomes unimportant. The subroutine may read different data items on the stack by using the indexed addressing modes with the stack pointer as the base register. Complex subroutines may require that the last word placed on the stack (the word with the lowest virtual address, because the stack expands toward low addresses) contain the number of parameters passed so that the program does not use other data also on the stack but not intended as parameters.

Separate Stack and Index Pointers

Using the stack pointer as the base address for indexed addressing presents problems if the subroutine must, in turn, pass data to another subroutine. Each time the first subroutine calculates a parameter for the second subroutine, it pushes the parameter onto the stack. The address in the stack pointer changes to reflect the new data on the stack. As a result, all instructions in the first subroutine which contain index constants are invalid, because the base value that the index constants are supposed to modify has changed. It would be very difficult, if not impossible, to write a subroutine that could use different index constants as the stack pointer changes (because to remain reentrant, the program cannot change any part of the instruction code). A much simpler solution is to separate the base register from the stack pointer by copying the stack pointer value into another general register before using the stack for any other data. This is still reentrant because any change of virtual machine also changes the contents of (or the selection of) all general registers.

The register commonly used as a separate index pointer is register 5. The standard method of calling subroutines in reentrant programs uses register 5 as the index pointer, register 6 as the stack pointer, and a word on the stack (at the address contained in the index pointer) that indicates the number of parameters on the stack. In addition to providing a straightforward and completely reentrant structure, this method is completely compatible with a similar form of non-reentrant subroutine call. The same subroutine can be called both by reentrant programs and by simpler programs that are non-reentrant.

Subroutine Call Compatibility

In a non-reentrant program, the parameters passed to a subroutine are placed in-line; i.e., they are in the addresses immediately following the address of the calling instruction. The subroutine call and return instructions use a register to store the program counter value for the calling program; the value in the program counter at the time the subroutine call (jump to subroutine or JSR) instruction is executed is the address of the word following the JSR instruction. The standard register specified in the JSR instructions is register 5; register 5 can be used as an index pointer while the stack is used for data storage during the execution of the subroutine. The JSR instruction does not destroy the previous contents of register 5 when it stores the return address in that register; the previous contents are pushed on the stack, and are automatically restored by a Return from Subroutine (RTS) instruction.

When the RTS instruction restores the Program Counter (PC) value stored by the JSR instruction, the calling program must have some means of bypassing the stored data to get to the next instruction. The word immediately following the calling instruction must contain the number of words occupied by the parameters. Both of these requirements can be fulfilled by placing a branch instruction in the return location; the branch instruction advances the PC so that the first word after the line parameters, and the offset in the eight least-significant bits of the branch instruction, contain the number of words used for the parameters (the offset is multiplied by 2, before use, to generate a byte address).

The calling sequence and in-line parameter structure used by non-reentrant routines permits the subroutine to return control to the calling routine with an RTS R5 instruction. For compatibility, the reentrant subroutine call must also permit the same RTS R5 instruction to perform the return. However, when a subroutine has been called in a reentrant manner, R5 points to a location on the hardware stack, not to the calling program. In addition, the space in the stack area used by the subroutine call must be released (the stack pointer must be adjusted to point to the first location after the parameter area) so that any additional information on the stack (such as a return linkage to a routine that called the routine that called the current subroutine) is accessible. Thus, the word pointed to by R5 should contain an instruction, whose least-significant bits are the number of parameters passed to the subroutine, which can adjust the stack pointer and also complete the subroutine return sequence.

The MARK instruction performs this function in the PDP-11/70. A detailed description of the use of this instruction is contained in the *PDP-11/70 Processor Handbook*.

SECTION II

PROCESSOR

Unless otherwise indicated, references within this section pertain to this section only.

SECTION II PROCESSOR
CONTENTS

SECTION II PROCESSOR
CONTENTS (Cont)

SECTION II PROCESSOR
CONTENTS (Cont)

Page

CONTENTS (Cont)

Page

CHAPTER 5 DATA TRANSFERS

CHAPTER 6 ABORTS, TRAPS AND INTERRUPTS

SECTION II PROCESSOR
CONTENTS (Cont)

ILLUSTRATIONS

SECTION II PROCESSOR
ILLUSTRATIONS (Cont)

TABLES

SECTION II PROCESSOR
TABLES (Cont)

INTRODUCTION

The KB11-C Processor System is capable of manipulating, storing, and routing data. The processor is the system component that manipulates the data. Although the processor is designed to effect complicated changes to the data that it receives, it actually consists of elements making only simple changes. The complex data manipulation are acheived by combining a number of these simple changes in a variety of ways.

The processor consists of logical elements, each element designed to perform a specific function. For example, some elements store data, some read data from another part of the computer, and others perform simple modifying functions such as complementing the data or combining two operands, either by arithmetic or by logical means. These simple basic operations are combined into functional groups known as instructions. An instruction can include a number of operations so that data can be combined, changed, moved, or deleted. Instructions can be further combined into programs which use a number of instructions to construct even more complex operations.

The basic logical elements of the processor can perform only a small number of operations at one time. Therefore, to combine a number of these operations into an instruction, the instruction must be divided into a series of sequential steps. These steps are called machine states, or cycles, and may perform either a single operation or several operations at the same time. An instruction thus becomes a sequence of machine states. This sequence may be fixed or may provide alternate paths (branches); in the latter case, internal conditions determine which branch the instruction will follow.

The processor can be divided into several functional parts:

1. The *Interface* section exchanges data with devices external to the processor (Chapters 5 and 6).

2. The *Data Paths* section performs data handling functions (Chapter 2).

3. *Control* section includes the logic that determines which operations are to be performed during a particular state and what the next machine state should be (Chapter 1).

4. The *Timing* section generates clock signals which synchronize the various operations of the KB11-C Processor System (Chapter 4).

5. The *Control Registers* store the results of processor operations. This data may be used in determining future processor operations (Chapter 3).

The Interface section consists basically of logic necessary for transferring data between the processor, the Unibus, the memory, and the Console. The Data Paths and Control sections interact to perform the three main processor functions of data storage, modification, and routing.

The Data Paths section consists of storage registers, shift registers, multiplexers, and an Arithmetic Logic Unit (ALU). The multiplexers control the data flow between registers. The ALU executes the more complex data manipulations, while the shift registers move the data bits stored in them, either to the left or to the right.

Operation of the elements of the Data Paths section is determined by the Control section. Refer to Figure I-1. This section consists of a Read Only Memory (ROM) and its associated logic. The ROM contains 256_{10} (400_8) locations. Each location contains 68_{10} bits. This 64-bit ROM output is divided into 32 groups or fields, each of which controls a discrete part of the KB11-C Processor. One of these fields is called the Address Field (UADR or UAD). The UAD field from the current machine state is combined with selected data from other sections of the KB11-C Processor in the ROM address logic, whose output is the ROM address for the next machine state. In this manner, the required machine states are generated in the proper sequence. The UAD field may either be used as the next ROM address, or may be modified by the feedback from the other sections of the processor to generate the next ROM address. This allows for instruction branching that is dependent on other conditions, and also for the use of machine states that are common to several instructions. An auxiliary ROM in Memory Management uses the same address as the processor ROM.

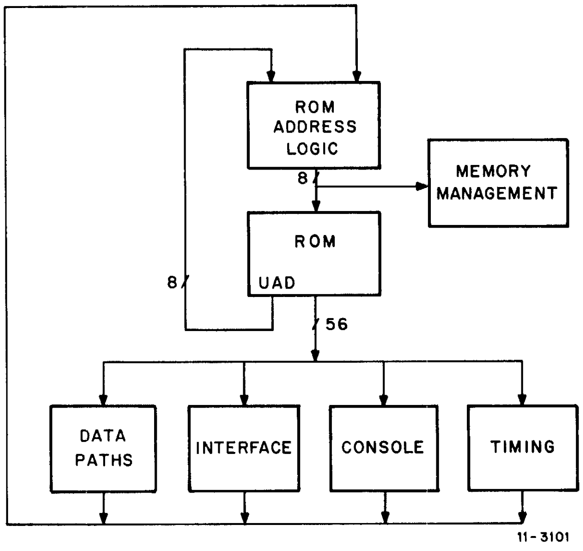


Figure I-1 Processor Control Section

CHAPTER 1
INSTRUCTION DECODE AND MICROPROGRAM CONTROL

The main function of the processor is to execute a *Program*, or sequence of *Instructions*.

Instructions are stored in memory. A *Program Counter* stores the address of the next instruction. At the end of the execution of one instruction, the processor fetches (reads from memory) the instruction that is to be processed next.

Instructions consist of a series of steps, called *Machine States*, or cycles, that are executed sequentially. This sequence of steps is unique to each instruction, although some steps, or series of steps, may be common to several instructions.

The sequence of operations within each instruction in the KB11-C is controlled by the microprogram Read Only Memory (ROM).

A ROM is a storage device whose contents are predetermined and cannot be changed. Each address generates a unique output. The KB11-C ROM has an 8-bit address, which allows 256_{10} different outputs, each consisting of 68 bits.

This 68-bit output (ROM word) is divided into 32 fields, each of which controls a different part of the processor.

The ROM word contains an address field, which in most cases is the address of the next ROM word: the ROM is self-sequencing. This address field can be modified by conditions internal or external to the processor, such as the instruction operation code, the addressing mode or other factors.

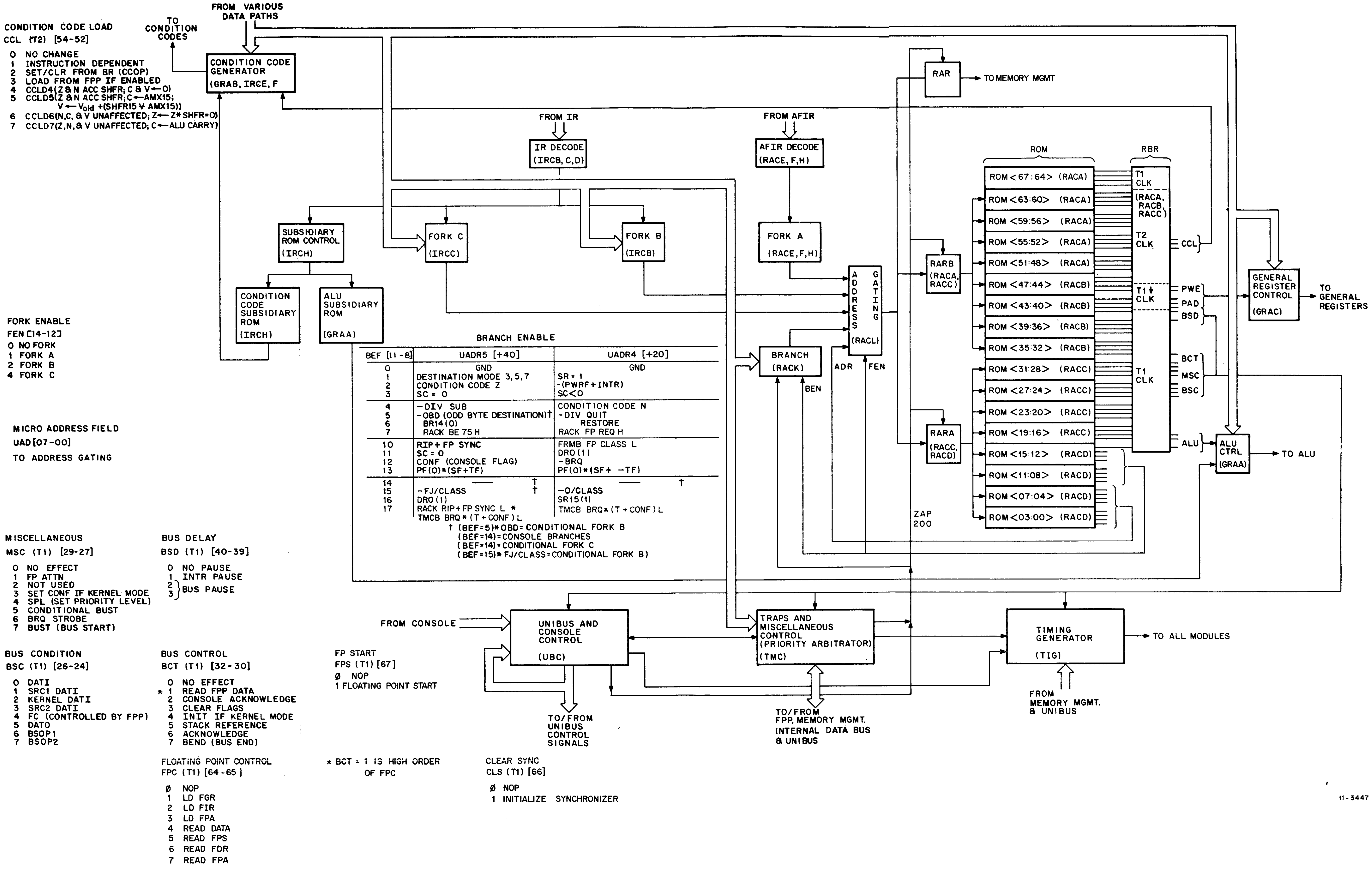
When an instruction is fetched (read from memory) it is stored in two instruction registers (IR):IRCA-IR(15:00) and RACJ AFIR (15:00) and in the FPP's FIRA if this option is installed. The contents of these registers are decoded, and these decoded outputs control the ROM address, along with inputs from other processor circuits.

The decoded outputs of the IR are also used to determine how the results of the executed instruction are interpreted in setting the *Condition Codes*. Refer to Paragraph 1.5.

BLOCK DIAGRAM

Figure 1-1 is a block diagram of the KB11-C Read Only Memory (ROM). The ROM contains 256_{10} or 400_8 processor control words. For each processor machine cycle, one of these stored words is output to the Data Paths section and to the other processor circuits. The ROM word is divided into fields, and each field controls a specific register, multiplexer or process of the processor. In Figure 1-1, each control field is listed by a mnemonic name and by bits of the microprogram word occupied by the control field. The control selection that is made, or the action that takes place for each value that can be stored in the field, is listed under the field name. Where possible, the field name and description are placed next to the logical element controlled by that field.

The microprogram ROM outputs that control other parts of the processor must be stored in a buffer register, so that the next microprogram word can be selected while the current word is being used. Therefore, a ROM Buffer Register (RBR) is provided for these outputs (Paragraph 1.1).



CONDITION CODE LOAD

CCL (T2) [54-52]

0 NO CHANGE
1 INSTRUCTION DEPENDENT
2 SET/CLR FROM BR (CCOP)
3 LOAD FROM FPP IF ENABLED
4 CCLD4(Z & N ACC SHFR; C & V ← O)
5 CCLD5(Z & N ACC SHFR; C ← AMX15;
$V \leftarrow V_{old} + (SHFR15 \forall AMX15))$
6 CCLD6(N,C, & V UNAFFECTED; Z ← Z* SHFR=O)
7 CCLD7(Z,N, & V UNAFFECTED; C ← ALU CARRY)

FORK ENABLE

FEN [14-12]

0 NO FORK
1 FORK A
2 FORK B
4 FORK C

MICRO ADDRESS FIELD

UAD [07-00]

TO ADDRESS GATING

BRANCH ENABLE

BEF [11-8]	UADR5 [+40]	UADR4 [+20]
0	GND	GND
1	DESTINATION MODE 3, 5, 7	SR = 1
2	CONDITION CODE Z	-(PWRF + INTR)
3	SC = O	SC<O
4	-DIV SUB	CONDITION CODE N
5	-OBD (ODD BYTE DESTINATION)†	-DIV QUIT
6	BR14 (O)	RESTORE
7	RACK BE 75 H	RACK FP REQ H
10	RIP + FP SYNC	FRMB FP CLASS L
11	SC = O	DRO (1)
12	CONF (CONSOLE FLAG)	-BRQ
13	PF(O)*(SF+TF)	PF(O)*(SF+ −TF)
14	—— †	—— †
15	-FJ/CLASS †	-O/CLASS
16	DRO (1)	SR15 (1)
17	RACK RIP + FP SYNC L * TMCB BRQ * (T + CONF) L	TMCB BRQ* (T + CONF) L

† (BEF=5)* OBD= CONDITIONAL FORK B
(BEF=14)= CONSOLE BRANCHES
(BEF=14)= CONDITIONAL FORK C
(BEF=15)* FJ/CLASS= CONDITIONAL FORK B)

MISCELLANEOUS

MSC (T1) [29-27]

0 NO EFFECT
1 FP ATTN
2 NOT USED
3 SET CONF IF KERNEL MODE
4 SPL (SET PRIORITY LEVEL)
5 CONDITIONAL BUST
6 BRQ STROBE
7 BUST (BUS START)

BUS CONDITION

BSC (T1) [26-24]

0 DATI
1 SRC1 DATI
2 KERNEL DATI
3 SRC2 DATI
4 FC (CONTROLLED BY FPP)
5 DATO
6 BSOP1
7 BSOP2

BUS DELAY

BSD (T1) [40-39]

0 NO PAUSE
1 INTR PAUSE
2, 3 } BUS PAUSE

BUS CONTROL

BCT (T1) [32-30]

0 NO EFFECT
* 1 READ FPP DATA
2 CONSOLE ACKNOWLEDGE
3 CLEAR FLAGS
4 INIT IF KERNEL MODE
5 STACK REFERENCE
6 ACKNOWLEDGE
7 BEND (BUS END)

FLOATING POINT CONTROL

FPC (T1) [64-65]

Ø NOP
1 LD FGR
2 LD FIR
3 LD FPA
4 READ DATA
5 READ FPS
6 READ FDR
7 READ FPA

FP START

FPS (T1) [67]

Ø NOP
1 FLOATING POINT START

* BCT = 1 IS HIGH ORDER OF FPC

CLEAR SYNC

CLS (T1) [66]

Ø NOP
1 INITIALIZE SYNCHRONIZER

Figure 1-1 Block Diagram

Three output fields are used to select the next microprogram word (FEN, BEF, and UAD). They are not buffered because they are used immediately and the resulting address is buffered. Immediately after the beginning of a machine cycle, when a new microprogram word becomes available, the ROM address generation circuits begin the calculation of the next ROM address. This corresponds to selecting the next machine state. The generated address is assembled by the address gating logic and loaded into the ROM Address Register (RAR). There are three copies of the RAR to accommodate the output loading required for 16 ROM elements, and to transmit the ROM address to Memory Management. (Refer to Paragraph 1.4.1.)

The address gating logic assembles the address from five sets of inputs. The basic input, which is always present, is the Address (UAD) field of the current microprogram word. The UAD is ORed with the outputs of the Branch logic, which is controlled by the BEF field of the microprogram word. The Branch Control logic selects a set of condition inputs from signals received from the processor data paths, the condition codes, and from the processor interface modules. Depending on the state of the selected inputs, the Branch Control generates one or two signals that are used to modify the address (Paragraph 1.4.4).

The three other inputs to the address gating circuits are from the Fork logic. The three forks are similar in implementation and purpose. Each fork uses combinational logic to decode the instruction type and a variety of processor conditions, and generates one of a number of addresses that is combined with the UAD input by masking. Each fork can be enabled by one bit in the Fork-ENable (FEN) microprogram field; normally all forks are disabled. No more than one fork is ever enabled at a time (Paragraphs 1.4.6 – 1.4.8).

The A Fork logic, used to select the machine state that follows an instruction fetch, requires a separate instruction register (AFIR) because this fork must operate rapidly and therefore puts a heavy load on the IR outputs. The B and C Forks decode inputs from the primary IR and use the outputs of a subsidiary ROM, which decodes some classes of instructions. These forks are used after a destination operand fetch and a source operand fetch, respectively.

To summarize the operation of the microprogram control logic: during each machine cycle, an address is assembled from any enabled fork combined with the address field of the microprogram word and any enabled branches. This address is loaded into the ROM address register to select a new microprogram word. At the beginning of the next machine cycle, the new microprogram word is loaded into the ROM buffer register and the sequence is continued.

On power-up, the ROM is initialized and the program is forced to a fixed address in memory which contains the power-up subroutine. This subroutine typically restores the program parameters that were stored during power-down. Refer to Chapter 6 (Traps, Aborts and Interrupts) for a description of these features.

DOCUMENTS

The documents listed below contain the information required to follow an instruction from fetch to execution.

1. KB11-C Flow Diagrams, drawing number D-FD-KB11-C-1, sheets 1 – 15. This set contains a block diagram of the processor on sheet 1, and the sequence of microprogram cycles in flowchart form, on sheets 2 – 15. The flowchart sheets are labelled "FLOWS 1" through "FLOWS 15", and are referred to in this manner throughout this manual. (Refer to Paragraph 1.2.)

2. ROM Map, sheets 12 – 15 of the RAC module schematic, drawing number D-CS-M8123-0-1. These four sheets reproduce the computer listing, in numerical order, of the contents of each ROM word, the name of each state, and the page of the Flows on which this state is shown. Refer to Paragraph 1.3.

The ROM and its control logic is shown on drawing D-CS-M8123-0-1, ROM & ROM Control (RAC module), and on drawing D-CS-M8132-0-1, IR Decode & Cond. Codes (IRC module).

1. The ROM, ROM Buffer Register (RBR) and ROM Address Register (RAR) are shown on sheets 2 - 5 of RAC (drawings RACA-RACD). Refer to Paragraphs 1.1 and 1.4.1.

2. The ROM Address bits (RADR), which are the inputs to the RAR are shown on sheet 11 of RAC (drawing RACL). Refer to Paragraph 1.4.2.

3. The Branch Control logic is on sheet 10 (RACK) of RAC. Refer to Paragraph 1.4.4.

4. The A Fork logic is shown on sheets 6 - 8 of RAC (drawings RACE, RACF and RACH). Refer to Paragraph 1.4.6.

5. The B Fork logic is on sheet 3 of IRC (IRCB). Refer to Paragraph 1.4.7.

6. The C Fork logic is on sheet 4 of IRC (IRCC). Refer to Paragraph 1.4.8.

7. The Condition Code logic is on sheets 6 through 9 of IRC (IRCE - IRCJ). Refer to Paragraph 1.5.

1.1 MICROPROGRAM ROM AND BUFFER REGISTER

All control signals that are dependent only on the machine state (i.e., that are not dependent on asynchronous signals or on data inputs) are derived directly from the outputs of the microprogram ROM. The ROM contains 256 68-bit words; during each processor cycle, one word is fetched from the ROM and stored in a buffer register. The outputs of the buffer register are transmitted to the other modules of the processor to act as control signals or to be used in combinational logic that generates control signals for all processor operations.

The ROM is implemented by 16 256-word × 4-bit read-only memories.

The buffer register is implemented primarily by 74S174 D-type hex flip-flop registers. (Some bits are implemented by individual flip-flops to provide separate input clocking or greater output load capacity.)

Various ROM bits are clocked into the output buffer register at different times. Most bits are clocked by the T1 pulse, while others are clocked by the T2 pulse. Certain bits are clocked on the trailing edge of the T1 pulse to allow slightly more time for the processor to complete operations started by the previous machine cycle.

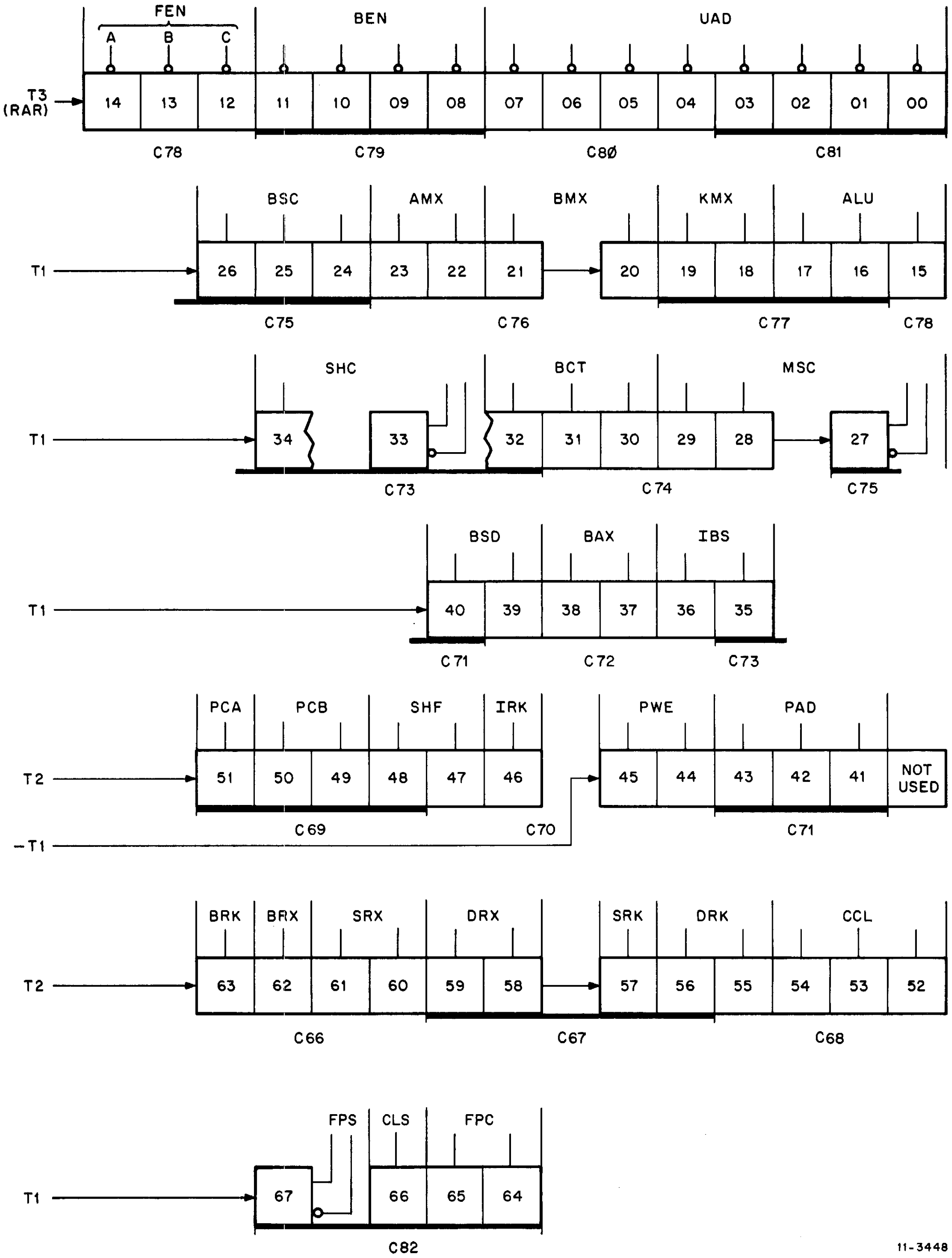
Figure 1-2 shows the ROM output bits, the type of ROM IC that generates each bit (i.e., C71), which groups of bits are stored in one 6-bit IC register, and the time at which they are clocked into the RBR. Table 1-1 gives much of the same information, plus the name given to each field.

The output buffer register, shown on drawing RACA, is clocked by the T2 pulse; none of the control signals transmitted from the 18 bits of storage on this drawing can be assumed to have settled before the T3 pulse.

Five output signals are derived from the contents of the buffer register that is clocked by the falling edge of the T1 pulse, rather than the leading edge (drawing RACB). These signals (two pad write-enable and three pad address lines) gate the writing of information into the processor general registers. The data is transferred into the registers by writing them with the T1 pulse, so these enable signals must not change until after the T1 pulse has occurred.

One of the 6-bit output registers, shown on drawing RACC, stores the output of bit 34 and of bits 32 - 28 of the ROM. Bit 33 is stored in a separate flip-flop. This permits the buffer register to transmit both polarities of USHC00, with no additional signal delays. Bit 27 of the ROM, which generates UMSC00, is also stored on a separate flip-flop to generate both polarities.

The microprogram bits which are used to calculate the new ROM address are used only on the RAC module, so they are not brought to module pins. However, several of the branch-enable signals are required either in both polarities or with greater fan-out capacity; UBEF03, UBEF01, and UBEF00 are buffered by more than one gate.



NOTE:
C82 = ROM IC type
Each 6-bit group: one 74S174 register, except bits 66-64 which are clocked into a 74S175 register.
Bits 27,33,67 are individual 74S74 flip-flops.

Figure 1-2 ROM Word: Clock, ICs and Registers

Table 1-1
Microprogram Bit Usage

Bit Positions	Contents	Clocked At
	RACA	
67	FP start (UFPS)	T1
66	clear sync (UCLS)	T1
65-64	Floating Point Control (UFPC)	T1
63	bus register clock (UBRK)	T2
62	bus register multiplexer (UBRX)	T2
61–60	source register MUX (USRX)	T2
59–58	destination register MUX (UDRX)	T2
57	source register clock (USRK)	T2
56–55	destination register clock (UDRK)	T2
54–52	condition-code load (UCCL)	T2
51	program counter A CLK (UPCA)	T2
50–49	program counter B CLK (UPCB)	T2
48–47	shifter control (USHF)	T2
46	instruction register CLK (UIRK)	T2
	RACB	
45–44	pad write-enable (UPWE)	T1 + 15 ns
43–41	scratchpad address (UPAD)	T1 + 15 ns
40–39	bus delay (UBSD)	T1
38–37	bus address multiplexer (UBAX)	T1
36–35	internal bus (UIBS)	T1
	RACC	
34–33	shift counter (USHC)	T1
32–30	bus control (UBCT)	T1
29–27	miscellaneous control (UMSC)	T1
26–24	bus conditions (UBSC)	T1
23–22	A multiplexer (UAMX)	T1
21–20	B multiplexer (UBMX)	T1
19–18	constant multiplexers (UKMX)	T1
17–15	arithmetic logic unit cont (UALU)	T1
	RACD	
14	fork C enable (UCFEN)	not buffered
13	fork B enable (UBFEN)	not buffered
12	fork A enable (UAFEN)	not buffered
11–08	branch-enable (UBEF)	not buffered
07–00	microprogram address (UADR)	not buffered

1.2 FLOW DIAGRAMS

The Flows are a description, in flowchart form, of the operation of the KB11-C Processor. Refer to Figure 1-3. Each cycle, or machine state, is represented on the Flows by a rectangular box. The top part of this box describes the operations executed during the cycle. The bottom part lists the actual operations that occur at each timing pulse.

The following information is supplied to aid in understanding and using the Flows:

1. A note on timing (Paragraph 1.2.1).
2. A glossary of abbreviations and terms used on the Flows (Paragraph 1.2.2).
3. A definition of Instruction Classes (Paragraph 1.2.3).
4. A description of Addressing Modes as they relate to operand fetch (Paragraph 1.2.4).
5. A description of the Flow Diagrams, page by page, which explains in general terms the use of the cycles on each page (Paragraph 1.2.5).
6. Tables listing the cycles on each Fork used by each instruction (Paragraph 1.2.6).

1.2.1 ROM Timing

Refer to Figure 1-4. The ROM address RACL RADR(07:00) H is clocked into the ROM address register at T3. The ROM output for the new cycle is clocked into the RBR at T1 – T2.

NOTE

The KB11-C is controlled by the clock circuits described in Chapter 4, Timing Generator. For the purposes of this Chapter and of Chapters 2 and 3, it must be known that there are two types of clock signals: the timing pulses, T1 – T5 and the time states, TS1 – TS5. The timing pulses are 15 ns wide and occur at 30 ns intervals. The time states occur at the same time as the timing pulse of the same number (TS1 occurs at the same time as T1) and are asserted for 60 ns.

The timing pulse shown as "T6" on the Flows occurs at T1 of the next cycle.

1.2.2 Glossary

The symbols, abbreviations and terms listed below occur on the Flow Diagrams and are also used in the text of this manual.

SYMBOLS

(OP CODE).B – Refers to both the word and byte instructions, when describing instruction classes, e.g.: "NEG.B" means "NEG and NEGB."

+ is used for a logical inclusive OR.

* is used for a logical AND.

ANGLE BRACKETS ⟨...⟩ – Indicates operations that are executed for diagnostic purposes only and are not necessary to the operation performed by the cycle.

$ – Instruction dependent. See Chapter 2.

ACKN – ACKNowledge: signal that clears certain trap and abort flags when they have been serviced.

AFIR – See IR

ALU – Arithmetic Logic Unit. See Chapter 2.

BA – Bus Address: Example: BA–PCB means that the PCB is used as the address for a data transfer.

BC – Bus Condition: defines the type of data transfer that is to be executed; example: BC–DATI

BEND – Bus END: aborts a data transfer cycle which cannot be completed because of an abort condition (refer to Chapter 6) or one which was started in the previous cycle and which is not required. See Chapter 5.

BR – Bus Register: stores data received during data transfers; also used as temporary storage during instruction execution.

BRQ STROBE – Signal which clocks traps and interrupts into the request register. See Chapter 6.

BUS – Source of data during any data transfer: may be Unibus, Internal Bus or Cache; example: BR–BUS.

BUS PAUSE – Second ROM state of any data transfer. See Chapter 5.

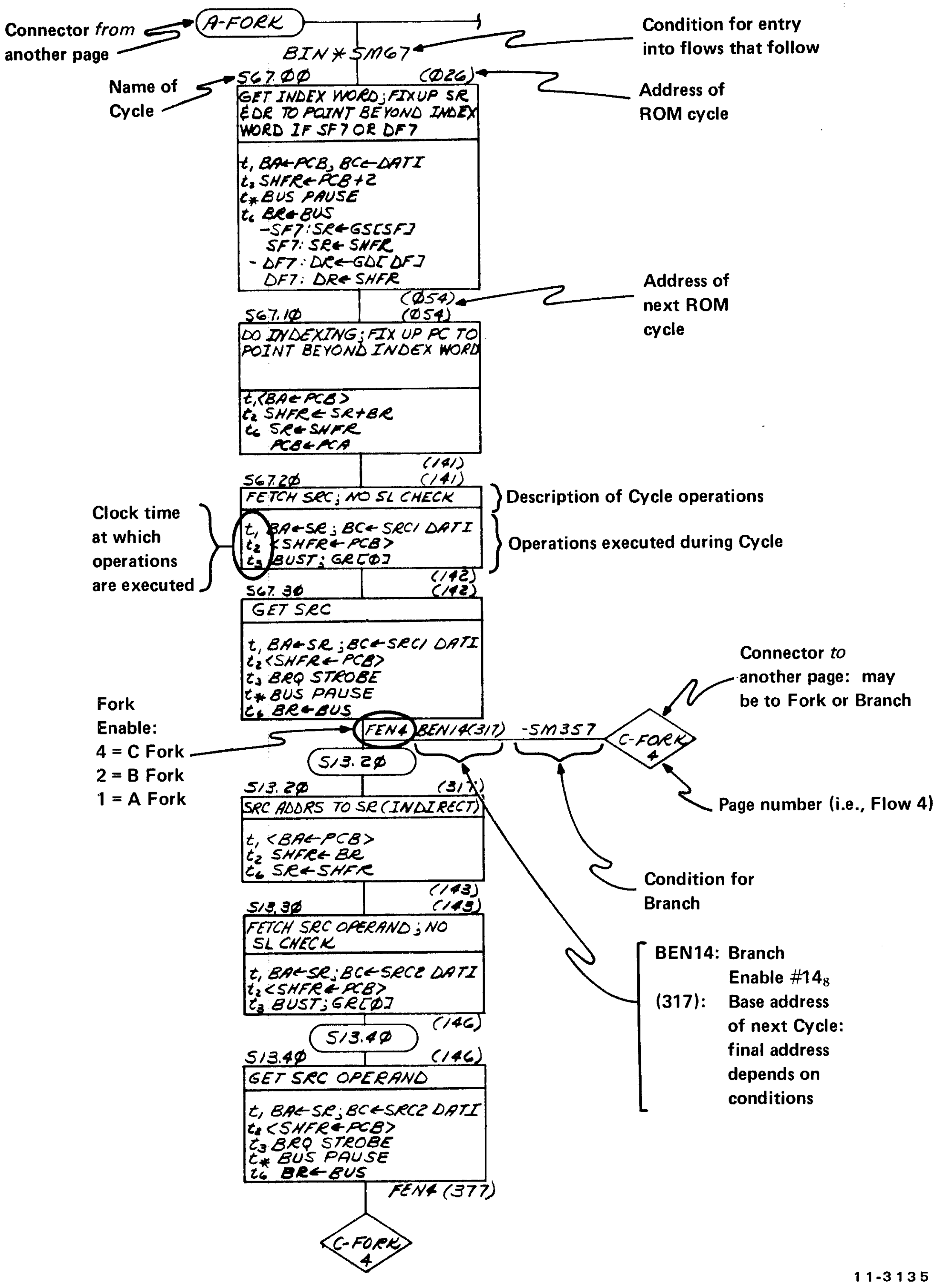


Figure 1-3 Flow Chart Symbols (P/O Flows 2)

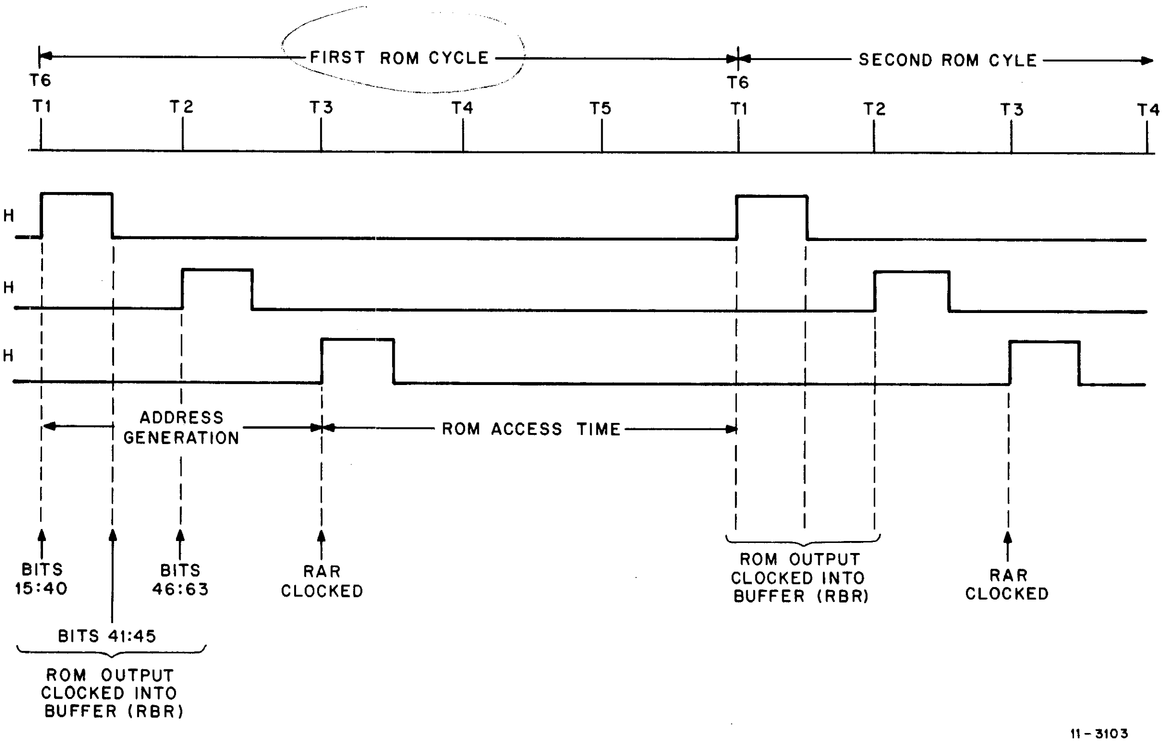


Figure 1-4 ROM Timing

BUST – BU STart: first cycle of any data transfer. See Chapter 5.

BXX DISP – The left shifted (multiplied by 2) and sign extended value of the displacement field of a branch instruction.

CC – Condition Codes

CCLD – Condition Code Load

CHECK STACK LIMIT – The contents of GD[6] are checked to see if there is a stack violation. See Chapter 6.

CLEAR FLAGS – Asserted when UBCT=3: clears the Address and Stack Error Flags. See Chapter 6.

CONF – CONsole Flag: causes the processor to halt when set.

DATI – Transfer of one word of data to the processor from memory or from a Unibus device. SRC1, SRC2, KERNEL DATI. See Chapter 5.

DATO – Transfer of one word of data from the processor to memory or to a Unibus device.

DF – Destination Field: bits 02:00 of instruction word: this number is the address of a register.

DM – Destination Mode: bits 05:03 of instruction word.

DR – Destination Register: see Chapter 2.

EALU – Floating Point Processor (FPP) ALU.

FC – FPP C1 line.

FCC – FPP Condition Codes.

FDR – FPP Data Register.

FIRA – FPP Instruction Register.

FPA – FPP Address Register

FP ATTEN – Signals the FPP that data transfer is complete.

FP READ DATA – Processor request for FPP data.

FPS – FPP Status Register.

FP START – Processor signal to FPP to initiate operation.

GD[X] - General Destination register. See Chapter 2. "X" designates the register number, e.g.: GD[4]; GD[DF] is the register designated by the Destination Field of the instruction word. The notation "GD[X]" means that the register is read.

GR[X] - General Register: includes both GD and GS when writing into these registers.

GS[X] - General Source Register. See Chapter 2. "X" designates the register number, e.g.: GS[4]; GS[SF] is the register designated by the Source Field of the instruction word. The notation "GS[X]" means that the register is read.

INIT - INITialization pulse (10 ms).

INTR PAUSE - INTerRupt PAUSE: the processor stops and accepts an interrupt vector from the Unibus. See Chapter 6.

IR,AFIR - Instruction Register which stores the instructon word.

Left Arrow (←) - Signifies transfer of data to unit on left from unit on right; example: BR←BUS, the BR receives data from the BUS.

PC,PCA,PCB - Program Counter. See Chapter 2.

SC - Shift Counter. See Chapter 2.

SF - Source Field: bits 08:06 of Binary instruction word; this number is the address of a register.

SHFR - SHiFteR. See Chapter 2.

SM - Source Mode: bits 11:09 of binary instruction word.

SR - Source Register. See Chapter 2.

SRCCON - Value generated to modify the SR during auto increment or decrement addressing mode.

SV - Start Vector: address of a word that contains the address that is entered on power-up. See Chapter 6.

SWAP(XX) - The SHFR moves the low byte into the high byte position and the high byte into the low byte position of the designated register.

TV - Trap Vector: address of a word that contains the address of a subroutine that is entered after a trap. See Chapter 6.

1.2.3 Instruction Classes

The instructions in the PDP-11 Instruction Set are divided into classes by the decoding logic on RAC and IRC. Some of these classes are used on the Flows to determine the machine state to which an instruction will go next.

During BSOP1 and BSOP2 data transfer cycles, one of several types of bus cycles (DATI, DATIP, DATO or DATOB) may be executed during a given machine state. The type of bus cycle that is executed during one of these machine states also depends on the instruction class. These instruction classes are described as follows:

P/CLASS - Defines a group of instructions which require a DATIP instead of a DATI cycle when obtaining the word which is to be operated on. This allows for modification of the word without requiring memory to restore the word first during a DATI and then again during a DATO. In addition, it provides an interlock, i.e., the location cannot be accessed by another device while it is being operated on. The following instructions are P/class:

00 03 DD	SWAB	07 4R DD	XOR
00 50 DD	CLR	10 50 DD	CLRB
00 51 DD	COM	10 51 DD	COMB
00 52 DD	INC	10 52 DD	INCB
00 53 DD	DEC	10 53 DD	DECB
00 54 DD	NEG	10 54 DD	NEGB
00 55 DD	ADC	10 55 DD	ADCB
00 56 DD	SBC	10 56 DD	SBCB
00 60 DD	ROR	10 60 DD	RORB
00 61 DD	ROL	10 61 DD	ROLB
00 62 DD	ASR	10 62 DD	ASRB
00 63 DD	ASL	10 63 DD	ASLB
00 67 DD	SXT	11 SS DD	MOVB
04 SS DD	BIC	14 SS DD	BICB
05 SS DD	BIS	15 SS DD	BISB
06 SS DD	ADD	16 SS DD	SUB

I/CLASS - Defines a class of instructions which require a DATI during a BSOP1:

00 57 DD	TS	07 1R SS	DIV
00 65 SS	MFPI	10 57 DD	TSTB
02 SS DD	CMP	10 65 SS	MFPD
03 SS DD	BIT	12 SS DD	CMPB
07 0R SS	MUL	13 SS DD	BITB

O/CLASS - Defines a class of instructions which require a DATO during a BSP1: 01 SS DD MOV and X0 66 DD MTP

BIN(ary) - All double-operand instructions; may require both source and destination calculations:

01 SS DD	MOV	11 SS DD	MOVB
02 SS DD	CMP	12 SS DD	CMPB
03 SS DD	BIT	13 SS DD	BITB
04 SS DD	BIC	14 SS DD	BICB
05 SS DD	BIS	15 SS DD	BISB
06 SS DD	ADD	16 SS DD	SUB

DAC - (Destination Address Calculation) All single-operand, register to destination or BIN*SM0 instructions:

always:

00 01 DD	JMP	07 1R SS	DIV
00 03 DD	SWAB	07 2R SS	ASH
00 4R DD	JSR	07 3R SS	ASHC
00 50 DD	CLR	07 4R DD	XOR
00 51 DD	COM	10 50 DD	CLRB
00 52 DD	INC	10 51 DD	COMB
00 53 DD	DEC	10 52 DD	INCB
00 54 D	NEG	10 53 DD	DECB
00 55 DD	ADC	10 54 DD	NEGB
00 56 DD	SBC	10 55 DD	ADCB
00 57 DD	TST	10 56 DD	SBCB
00 60 DD	RO	10 57 DD	TSTB
00 61 DD	ROL	10 60 DD	RORB
00 62 DD	ASR	10 61 DD	ROLB
00 63 DD	ASL	10 62 DD	ASRB
00 65 SS	MFPI	10 63 DD	ASLB
00 67 DD	SXT	10 65 SS	MFPD
07 0R SS	MUL		

if SM0:

01 SS DD	MOV	11 SS DD	MOVB
02 SS DD	CMP	12 SS DD	CMPB
03 SS DD	BIT	13 SS DD	BITB
04 SS DD	BIC	14 SS DD	BICB
05 S DD	BIS	15 SS DD	BISB
06 SS DD	ADD	16 SS DD	SUB

E/CLASS - (Execute class) No address calculation is required. These instructions use EXC.80 or EXC.90 (Flows 3). In general, these are DAC*DM0 or BIN*SM0*DM0:

00 03 DD	SWAB	06 SS DD	ADD
00 50 DD	CLR	07 4R DD	XOR
00 51 DD	COM	10 50 DD	CLRB
00 52 DD	INC	10 51 DD	COMB
00 53 DD	DEC	10 52 DD	INCB
00 54 DD	NEG	10 53 DD	DECB
00 55 DD	ADC	10 55 DD	ADCB
00 56 DD	SBC	10 56 DD	SBCB
00 57 DD	TST	10 57 DD	TSTB
00 60 DD	ROR	10 60 DD	RORB
00 61 DD	ROL	10 61 DD	ROLB
00 62 DD	ASR	10 62 DD	ASRB
00 63 DD	ASL	10 63 DD	ASLB
00 67 DD	SXT	12 SS DD	CMPB
01 SS DD	MOV	13 SS DD	BITB
02 SS DD	CMP	14 SS DD	BICB
03 SS DD	BIT	15 SS DD	BISB
04 SS DD	BIC	16 SS DD	SUB
05 SS DD	BIS		

BSOP1 - (BuS OPeration 1) When the ROM Bus Condition (UBSC) equals 6 during a bus cycle (data transfer), a DATO is executed for an O/class instruction, a DATIP for a P/class or a DATI if the instruction is neither O/class nor P/class. This condition is shown on the Flows as BC←BSOP1.

BSOP2 - (BuS OPeration 2) When UBSC=7 during a bus cycle, a DATOB is executed for a byte instruction and a DATO for a word instruction. This condition is shown on the Flows as BC←BSOP2.

J/CLASS - 00 01 DD JMP or 00 4R DD JSR -See FJ/class.

F/CLASS - Floating Point Processor instructions 17 XX XX - See FJ/class.

FJ/CLASS - F/class or J/class, which require one bus cycle less after the destination address calculation cycles than other DAC instructions (Flows 5 and 6).

1.2.4 Addressing Modes and Operand Fetch

In general, the following steps are required for the execution of an instruction:

1. Instruction Fetch: The instruction word is read from memory. The PCB is used as an address and a DATI is executed in FET.10. The instruction word is stored in the instruction registers (IR and AFIR).
2. Source Operand Fetch: This step is required only by BIN instructions whose source mode is not 0 (-SM0). This may require up to three DATI bus cycles, depending on the addressing mode (refer to Paragraphs 1.2.4.1 and 1.2.4.2).
3. Destination Operand Fetch: This step is required by all instructions that have a destination operand when the destination mode is not 0 (-DM0). Up to three bus cycles may be required, depending on the addressing mode. Address word fetches are DATIs; operand bus cycles may be DATIs (I/class instructions), DATOs or DATOBs (O/class) or DATIP/DATO(B)s (P/class).
4. Execution: After fetching the operand(s), the operation specified by the op code is performed. Execution may require several cycles or may be part of the destination operand fetch.

PDP-11 instructions allow six bits for each operand address. Three of these bits point to one of the general registers; the other three define one of eight addressing modes, 0 - 7, which are defined in Paragraphs 1.2.4.1 and 1.2.4.2. The position of the bits in the instruction word is shown in Figure 1-5. Unary, or single-operand instructions require only a destination (DST) address, located in bits 05:00. Binary, or double-operand instructions require both a source (SRC) and a destination address; the SRC is located in bits 11:06 and the DST in bits 05:00.

NOTE

In the KB11-C, those FPP instructions whose bits <11:06> = 0 are also classified as Mode 0 (CFCC, SETF, SETI, SETD, SETL, op codes 170000-170012). These are FPP Register operations. Refer to Paragraph 1.2.5.2.

The mode determines how the contents of the register are to be used. Addressing is said to be:

DIRECT - when the contents of the register are the operand (mode 0);

DEFERRED - when the contents of the register are the address of the operand or the address of the address of the operand (modes 1 - 5 and 7);

INDEXED - when the contents of the register are added to those of the word following the instruction to obtain the address of the operand (mode 6).

Mode 7 is indexed and deferred. Modes 4 and 5 decrement the contents of the register by 2 before address determination. Modes 2 and 3 increment the contents of the register by 2 after the address determination.

Up to three bus cycles are required to obtain each operand, one for each level of deferral, plus one for indexing.

NOTE

Programming documentation sometimes refers to the contents of bits 05:00 of an instruction word as a Source address. The KB11-C logic, however, treats any operand field in bits 05:00 as a Destination address. For example, MFPI and MFPD are shown on the PDP-11 Programming Card as 0065SS and 1065S, where "SS" indicates the source; these two instructions, however, are DAC and are executed as such: the contents of the SS field (bits 05:00) are used in the same manner as the bits 05:00 (=DD) in an INC (0052DD) instruction.

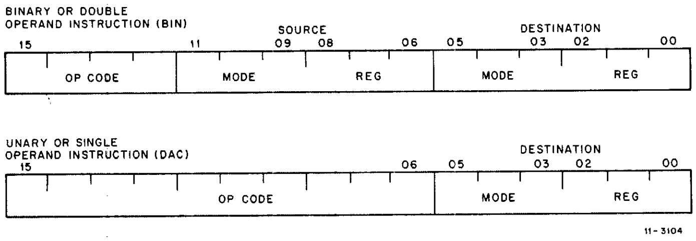


Figure 1-5 Source and Destination Mode Formats

1.2.4.1 General Register Addressing – "R" is any general register but register 7 (PC). The number of bus cycles listed below for each mode is that required for operand fetch.

Mode	Name	Definition
0	REGISTER Symbolic: %R	Register R contains the operand.
Example: CLR %3=005003		No bus cycle required.
1	REGISTER DEFERRED Symbolic: (R)	Register R contains the address of the operand.
Example: CLR (3)=005013		One bus cycle is required.
2	AUTO-INCREMENT Symbolic: (R)+	Register R contains the address of the operand. The register is incremented after the operand has been fetched.
Example: CLR (3)+=005023		One bus cycle required.
3	AUTO-INCREMENT DEFERRED Symbolic: @(R)+	Register R contains the address of a location which contains the address of the operand. The contents of the register are incremented after its use.
Example: CLR @(3)+=005033		Two bus cycles are required.
4	AUTO-DECREMENT Symbolic: -(R)	The contents of Register R are decremented, then used as the address of the operand.
Example: CLR -(3)=005043		One bus cycle is required.
5	AUTO-DECREMENT DEFERRED Symbolic: @-(R)	The contents of register R are decremented by 2. The register then contains the address of a location which contains the address of the operand.
Example: CLR @-(3)=005053		Two bus cycles are required.
6	INDEX Symbolic: X(R)	The contents of register R are added to the word X to which the PC is pointing. This sum is the address of the operand. The word to which the PC is pointing is called the INDEX word (engineering term) or BASE (programming term). This word may be the second or third word of an instruction.
Example: CLR 100(3)=005063 000100		Two bus cycles are required.

Mode	Name	Definition
7	INDEX DEFERRED Symbolic: @X(R)	Same as Mode 6, except that the sum is the address of a location which contains the address of the operand.

Example:

```
  CLR @100(3)=005073
              000100
```

Three bus cycles are required.

1.2.4.2 Program Counter Addressing – "R" is the PC (general register 7). The number of bus cycles listed below for each mode is that required for operand fetch.

NOTE

Modes 2, 3, 6 and 7 are also used with the PC as the register. The machine sequence for obtaining the operand is the same in this case as that used when any other register is used. Modes 0, 1, 4 and 5 are not illegal, but are of no practical use.

Mode	Name	Definition
2	IMMEDIATE Symbolic: #n	The PC, after the instruction fetch, contains the address of the operand, which is the word contained in the memory location following that in which the instruction word is stored. The PC is incremented by 2.

Example:

```
MOV #100,R0 ; MOVE 100(8) TO REGISTER 0
```

The operation of this mode is explained as follows:

The statement MOV #100,R0 assembles as two words. These are:

```
    0 1 2 7 0 0
    0 0 0 1 0 0
```

Just before this instruction is fetched and executed, the PC points to the first word of the instruction. The processor fetches the first word and increments the PC by two. The source operand mode is 27 (autoincrement the PC). Thus, the PC is used as a pointer to fetch the operand (the second word of the instruction) before being incremented by two, to point to the next instruction.

One bus cycle is required.

Mode	Name	Definition
3	ABSOLUTE Symbolic: @#A	Same as Mode 2, except that the word that follows the instruction is the address A of the operand, instead of the operand itself.

Example:

```
Example:  CLR @#100 = 005037
                      000100
```

Two bus cycles are required.

Mode	Name	Definition
6	RELATIVE Symbolic: A	Relative mode is assembled as index mode, using register 7, the PC, as the index register. The base of the address calculation, which is stored in the second or third word of the instruction, is not the address of the operand (as index mode), but the number which, when added to the PC, becomes the address of the operand. Thus, the base is X-PC, which is called an offset. The operation is explained as follows:

Example:

If the statement MOV 100,R3 is assembled at absolute location 20, the assembled code is:

```
Location 20:  0 1 6 7 0 3
Location 22:  0 0 0 0 5 4  (54 = 100-24)
```

Mode	Name	Definition
		The processor fetches the MOV instruction and adds two to the PC so that it points to location 22. The source operand mode is 67; that is, indexed by the PC. To pick up the base, the processor fetches the word pointed to by the PC and adds two to the PC. The PC now points to location 24. To calculate the address of the source operand, the base is added to the designated register. That is, BASE+PC=54+24=100, the operand address. Two bus cycles are required.
7	RELATIVE DEFFERED Symbolic: @A	Same as Mode 6, except that the sum BASE+PC is the address of a location which contains the address of the operand. Three bus cycles are required.

1.2.4.3 A and C Forks: Operand Fetch – After an instruction has been fetched and decoded, the operand(s) are obtained from memory, if the addressing mode is other than 0. The operation required by the operation code is then executed.

The A FORK is used by all instructions:

1. Binary instructions that require source mode calculation (-SM0) calculate their source address and fetch the source operand.

2. Binary instructions that require no source address calculation (SM0) and single-operand instructions are DAC and calculate the destination address and fetch the destination operand.

3. Binary instructions with both SM0 and DM0, single-operand instructions with DM0, and instructions that are not part of any one of the classes listed on Flows 3 and 5 are executed.

The C Fork is used by F/class instructions or by binary instructions with source mode other than 0 (-SM0) to calculate the destination address and to fetch the destination operand after the source operand has been obtained on the A FORK.

Figure 1-6 shows the A and C Fork source and destination calculation cycles. After the instruction has obtained its operand(s) on these forks, it is executed on the B Fork.

1.2.5 Flowchart Description

The KB11-C Processor flowcharts (drawing D-FD-KB11-C-1) are divided into 14 drawings that illustrate options of the flow. Where possible, a continuous sequence of machine states is shown on a single drawing. The succeeding paragraphs describe the machine operations illustrated on each drawing. The description does not attempt to give detailed information about each machine state shown on the drawing; this information can be derived directly from the flowcharts and the ROM map (Paragraph 1.3).

Data Transfers

Data transfers require two machine states: a preliminary or BUST cycle, which sets up the conditions for the PAUSE cycle, during which the data is transferred. Data transfers are described in detail in Chapter 5.

1.2.5.1 FLOWS 1

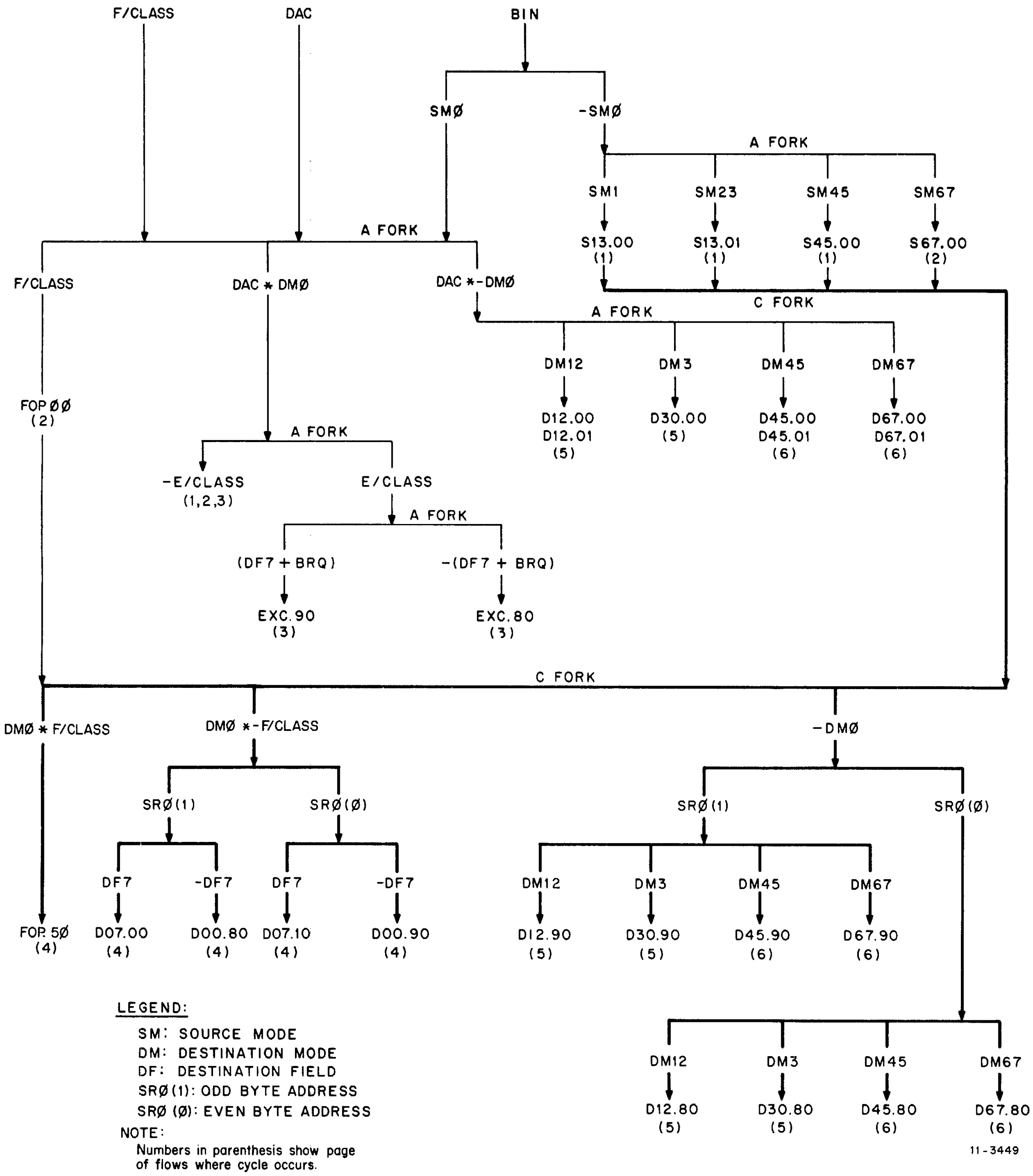
Instruction Fetch

Flows 1 illustrates the instruction fetch sequence, the address calculation sequence for five of the source modes, a special sequence for the MTPI and MTPD instructions, and the execution of the branch type instructions.

Fetch States

The basic instruction fetch sequence requires two machine states: FET.10 (fetch) and IRD.00 (IR decode). FET.10 completes a data transfer operation, begun during the last cycle of the previous instruction, which moves the instruction word from an external storage location to the instruction register (IR) and bus register (BR), and increments the program counter by 2. The instruction address is also stored in the FPA (FPA←BA), if the FP11-C option is present. If the data transfer is not overlapped (i.e., if the transfer was not begun before the end of the previous instruction), an additional state is required to begin the data transfer.

The additional state, FET.00, also checks for asynchronous operations (such as bus requests) that must be performed before beginning a new instruction, and branches to BRK.90 (break) if necessary. When the instruction fetch is overlapped, the machine state that begins the data transfer must also perform the same check.



LEGEND:

SM: SOURCE MODE
DM: DESTINATION MODE
DF: DESTINATION FIELD
SRØ(1): ODD BYTE ADDRESS
SRØ (Ø): EVEN BYTE ADDRESS

NOTE:
Numbers in parenthesis show page of flows where cycle occurs.

Figure 1-6 A and C Forks, General Case

Instruction Decoding

IRD.00 begins a new data transfer that fetches the word following the instruction word. This data transfer is used for address modes 6 or 7, and for fetching the next instruction whenever the instruction being executed does not require other data transfers.

In some cases, the CONDITIONAL BUST is not issued, i.e., when a data cycle is required but the PC, which is specified as the address in IRD.00, is not the required address. In this case, for example D30.00 (Flows 5), the DR is the address and a new BUST is issued. CONDITIONAL BUST, which is used only in IRD.00 (UMSC=5), and BUST are controlled by RACH BUST H. Refer to drawing RACH:

The four AND gates must be negated to assert BUST.

1. The top gate is negated when MCS=5 or 7.

2. The three other gates are enabled when MCS=5 (CONDITIONAL BUST in IRD.00).

3. The second gate from the top is asserted, and negates BUST during an IRD.00 that precedes S13.00 and S13.01 (BIN*SM123).

4. The third gate from the top is asserted, and negates BUST during IRD.00, if the instruction is a Branch and if there is a Brake Request (BRQ TRUE). FET.00, which is a BUST cycle, follows IRD.00 in this case.

5. The last gate prevents BUST from being asserted during an IRD.00, if this cycle precedes the three cycles that calculate destination modes 1, 2 and 3 on the A Fork (D12.00, D12.01, DAC*DM12; and D30.00, DAC*DM3; all on Flows 5). These cycles fetch the destination operand but use the DR as the address, instead of the PCB used by IRD.00.

   The NAND gate prevents the negation of BUST during IRD.00 when the cycle that follows it is S67.00 (BIN*SM67, Flows 2), if the destination mode of the instruction is 1, 2, or 3. This cycle gets the index word for source mode 6 and 7 of a binary instruction. The PCB is used here as the address and the bus cycle started in IRD.00 is completed. The NAND gate prevents BUST from being inhibited if the destination mode of the BIN instruction is 1, 2, or 3.

In other cases, this data transfer operation is aborted by a Bus End (BEND) operation in the machine state following IRD.00. During this machine state, the processor also loads the source and destination registers (SR and DR) with the contents of the general registers specified in the source and destination fields of the instruction; this operation is also done in anticipation of the use of this data, and in many cases the data loaded into the SR and DR is ignored. However, when the data is needed, the anticipatory transfers allow the processor to operate at maximum speed. The instruction word is stored in the FIRA (FIRA←BR), if the FP11-C option is installed.

Source Modes 1 – 5

The A Fork logic is enabled during IRD.00 (FEN 1), so the machine state that follows IRD.00 is determined by decoding the instruction and certain other conditions. Six of the possible sequences that follow IRD.00 are shown on Flows 1. These include the beginning of the data fetch sequence for all binary instructions that have a source mode of 1 – 5. If the source mode is 1, 2 or 3, the external data transfer is restarted with a new address and the incrementation of the source register is started for modes 2 or 3. If the source mode is 4 or 5, the external data transfer can not be started until the address has been decremented, so S45.00 performs a BEND. After performing the data transfer to fetch the word addressed by the source register, the sequence conditionally enables the C Fork logic. If the source mode is odd, another data transfer is required to fetch the data addressed by the word just fetched; otherwise the fork determines the next state.

Move to Previous Space Instructions

For an MTPI or MTPD (Move To Previous) instruction, MTP.00 and MTP.10 read an address from the stack pointer and begin a data transfer operation to fetch a data word that will be transferred to the destination address. The flow then transfers to the last state of the source-data-fetch sequence, because this state is alike for both the MTP sequence and the normal source data sequence.

Branch Instructions

For branch instructions, the A Fork logic determines whether the branch is successful, and if not, whether a bus request has been sensed. If the branch is successful, the PC must be changed before the next instruction is fetched; this is performed by the BXX.00 - BXX.05 (branch) machine state which aborts the previous data transfer. This state also strobes any new bus requests. The BRQ STROBE must be performed in the state preceding the state that starts the instruction fetch; this includes FET.10 (in case the A Fork logic returns control directly to FET.00), the next-to-last state of instructions that overlap the instruction fetch, and the last state of instructions that do not provide overlap. The machine state following BXX.00 is FET.00.

If the branch is not successful and no bus requests are sensed, the instrucion fetch continues the data transfer begun in IRD.00; if a bus request is sensed, the sequence returns to FET.00, which in turn transfers the sequence to BRK.00. Table 1-3E lists the ROM words used by each branch instruction for the four possible sequences.

1.2.5.2 FLOWS 2

Indexed Source Modes and Operate Instructions

Flows 2 illustrates the sequence of machine states for the data fetch for source modes 6 or 7, for the transfer of floating-point instructions to the FPP, and for the execution of five operate instructions.

Indexed Source Modes

For BIN*SM67, the indexed source modes for binary instructions, the transfer begun in IRD.00 is completed and an increment from the source register is added to the data word; the resulting data word is used for a second data transfer. When this transfer is complete, a conditional fork is used to transfer to the sequence required for the current instruction, unless an indirect-indexed address requires a third data transfer. In the latter case, the sequence continues through three machine states that are common to the sequences of all indirect source modes 3, 5, and 7, and in part to the MTPI or MTPD instruction.

Floating Point Instructions

When a floating-point instruction is recognized by the A Fork logic, the sequence is transferred to FOP.00 (floating-point operation). In this state, the contents of the Destination Register are stored in the BR; in the following state (FOP.10) the contents of the BR are stored in the FDR. Thus, at this point in the instruction execution, the instruction word, its address, and the contents of the General Register specified by the instruction are all stored in the FPP.

The instruction flow then goes to the C Fork logic to perform the address calculation:

1. For DM0 (which also includes FPP op codes 170000–170012, whose IR<11:06> = 0: CFCC, SETF, SETI, SETD and SETL), the next cycle is FOP.50 (Flows 4);

2. For -DM0, the FPP uses the same address calculation cycles as the processor instructions.

RTI and RTT Instructions

The RTI and RTT instructions differ only in the clocking of T bit traps after the data transfers, so the sequence of machine states is identical. This sequence performs two data transfers to restore the previous PC and PS words from the hardware stack, and performs two increment operations on the stack pointer. The sequence then continues with an instruction fetch.

RTS Instruction

The RTS sequence performs one register-to-register transfer and one external data transfer to restore the PC and the specified register, and updates the Stack Pointer (SP) after the transfer. The sequence then returns to the instruction fetch machine states.

SOB Instruction
The sequence of machine states for the SOB instruction first generates a new PC value, based on the offset in the instruction, and then restores the old PC value if the value in the specified register will be 0 after decrementing. This is done because the test on the value of the register requires one machine state in every case, which can be combined with the calculation of the new PC value, and because the branch is successful most of the time; thus, the extra machine state to perform the restoration of the old PC value is executed less often than if an extra state were required when the branch is successful. The SOB sequence initiates the fetch of the next instruction during the last machine state, which also performs the decrement on the specified register.

MARK Instruction
The machine state sequence for the MARK instruction transfers the contents of general register 5 to the PC, transfers the top word on the hardware stack to register 5, then begins fetching the next instruction. The operation of the MARK instruction assumes that the instruction has been fetched from the top of the hardware stack; for a discussion of the purpose and effects of the MARK instruction, see Chapter 4.

1.2.5.3 FLOWS 3

No Memory Reference Execution
Flows 3 illustrates the machine state sequences for a variety of instructions that do not require memory references other than the instruction fetch. A number of sequences are shown that transfer immediately to machine states on other pages; they are shown only to illustrate the routing from A Fork to these states. These sequences include the breakpoint trap (OP3), IOT trap, the EMT and TRAP traps, and several groups of reserved op codes, including OP7, OP22, and RSVD. The illegal instructions JMP or JSR, with destination mode 0, also transfer directly to a point in the trap sequence. The four instructions ASH, ASHC, MFPI, and MFPD are shown on other pages which do not show the A Fork flow line; therefore, off-page connectors are shown on this drawing for these instructions with destination mode 0 (for other destination modes of these instructions, the sequence transfers to the destination address calculation sequences shown on Flows 5 and 6).

Multiply and Divide with Destination Mode 0
For the multiply and divide instructions, a special sequence is used when the destination mode is 0. In either case, this sequence precedes the normal sequence for that instruction. MUL.80 (multiply) sets up the step counter and transfers to MUL.10, because MUL.00 is used to complete the data transfer begun in the destination data fetch sequence. In DVS.00 (divide start), the contents of the register specified for the destination operand are transferred to the BR, which corresponds to the result of the data fetch sequence for other destination modes.

E/Class and Negate Instructions
For the majority of instructions that operate on data, one machine state is required to perform the data manipulation. If both the source (if any) and destination modes are 0, the data is already in the SR and DR registers as a result of IRD.00. The data manipulation (selected by the subsidiary ROM for all except the NEG.B instruction) is performed, the data is stored in the general register specified by the destination field, and the sequence returns to the instruction fetch. The NEG and NEG.B instructions require two machine states because the complement and increment operations cannot be performed on the data during the same state; therefore the external data transfer operation started in IRD.00 is aborted (a bus operation cannot be carried across more than two machine states) and the sequence returns to FET.00. The other instructions complete the data operation and return to FET.10, unless a bus request has been sensed; because the transfer to the BRQ service sequence is performed by FET.00, the bus operation must be aborted.

RESET Instruction
Three processor control instructions, RESET, HALT and WAIT, are executed by sequences shown on this drawing. The RESET instruction transfers general register 0 to the DR so that the contents of R0 can be displayed in the DATA lights of the console during the reset operation, and then triggers the initialization pulse. The initialization is inhibited if the processor is not operating in the Kernel mode; in this case, the instruction is, in effect, a NOP. The machine state that triggers the pulse recycles to itself until the pulse (which lasts for 10 ms) is completed, and then returns the sequence to the instruction fetch sequence.

HALT Instruction

The HALT instruction does not actually stop the processor; instead, control is transferred to the console service sequence, which waits for manual intervention to determine further operations. This is performed by setting the console flag and then returning to the instruction fetch sequence where the console flag generates a BRQ, which in turn transfers to the break service sequence. The console flag is set only if the processor is in Kernel mode; a branch after HLT.10, (HALT) transfers control to the trap service sequence if the processor is not in Kernel mode, i.e., a HALT instruction in Super or User modes traps through location 4.

WAIT Instruction

The WAIT instruction is used to wait for an asynchronous condition that either initiates the execution of a service program or enters the console service sequence. The basic wait loop consists of two machine states, so that the BRQ STROBE in one state is available for the branch in the other state. When any BRQ is sensed, the sequence goes to the first of two states that test for console requests and then for interrupts or traps (other than T bit traps) that supply vectors. If neither is found, the sequence returns to the wait loop; otherwise, control is transferred to the appropriate sequence.

Processor Status Change Instructions

Two types of instructions that transfer data from the instruction word to the PS word are the CCOP instruction and the SPL instruction. The former affects only the condition code bits [PS(03:00)] and the latter affects only the priority bits [PS(07:05)]. In the CCOP instruction, the external data transfer begun by the IRD.00 state is aborted because the processor must maintain the data in the BR register until the PS word is reloaded. In the SPL instruction, the first state does the actual transfer to the priority. The second state also begins a new instruction fetch and control transfers to FET.10. SPL is a no-op (no change to the PS) unless the processor is in Kernel mode.

1.2.5.4 FLOWS 4

Destination Mode 0 Sequence

Flows 4 illustrates the five sequences used when the destination mode is 0. These sequences are entered through the C Fork microprogram address calculation; this fork is used to determine the next machine state after a source operand has been fetched. For all instructions except floating-point instructions, these sequences correspond to, or join, the sequences used when both the source and the destination modes are 0.

Not Register 7

When the destination specification in an instruction refers to any general register other than register 7 (the PC), and the other conditions for the sequences shown on this drawing are met, the instruction is executed by D00.90 (destination mode 0). If the source address is odd, a byte-swap operation must be performed on the contents of the BR before the instruction-dependent data manipulation operation. If the source mode is also 0, no byte swap is required, and the execution is performed by the EXC.8 (execute) machine state.

Register 7

When the destination register is 7, the PC is modified. Because the PC is stored as a separate register (not in the general register set), the execution is accomplished by EXC.90, which requires the source data to be in the SR register. A machine state is therefore required to transfer the source data from the BR to the SR. A byte swap can be combined with this transfer, if necessary.

Floating-Point Instructions

FOP.50 is the C Fork cycle used by all DM0*F/CLASS instructions, which include FPP Condition Code and accumulator to accumulator operations, as well as FPP writes to the processor general registers.

This sequence reads the FPP Status Register into the BR. If BRQ is true, a branch to FOP.60 is executed. In this cycle, the address of the FPP instruction is read into the BR; then, in FSV.90 (Flows 13), it is written back into PCA and PCB, and control is transferred to the service routine (BRK.00, Flows 12). The FPP instruction is aborted at this time and its address is saved. This same instruction will thus be fetched again and executed after the service routine.

FOP.30 repeats FOP.50 and waits for FP SYNC. If BRQ is true, control is transferred to the service routine as described above. If FP SYNC is received, FOP.40 is executed. FOP.30 cycles upon itself until either of these conditions is true.

FOP.45 instructs the FP11-C to execute the instruction (FP START).

1. In the case of a CFCC, the FPP Condition Codes are written to the PSW from the BR.

2. If the instruction requires a write into a processor General Register (FP REG WR), the data is read into the BR in FOP.65 then transferred to GR[DF] during FET.08, as the next instruction fetch is started.

3. If the instruction does not require a write into a processor general register, the instruction is done and control is transferred to FET.06.

1.2.5.5 FLOWS 5

Destination Modes 1 – 3

Flows 5 illustrates the machine state sequences used to fetch data specified by destination modes 1, 2, or 3. These sequences are entered from one of the two forks; some are entered from the A Fork decision point, for instructions which either do not require a source operand or have a source mode of 0, while others are entered from the C Fork decision point after the source operand has been fetched and placed in the SR.

Sequence Entry

All six sequences on this drawing start a data cycle (BUST). It should be noted that the CONDITIONAL BUST in IRD.00 is not asserted when the two A Fork sequences on Flows 5 are entered; this is because the PC is not the address required for the DM123 data cycles on this drawing.

The four sequences entered from the C Fork decision point also start by transferring the contents of the BR to the SR, so that the source data is available in both registers; the opposite transfer is performed for the A Fork entry to move the source data to the BR for the DATO that follows the destination address calculation. If the destination is 3, there is no point in loading the BR from the DR because the address fetched by the first external data transfer is stored in the BR for use in the next data transfer.

Destination Modes 1 and 2

There are two entries from the C Fork decision point for address modes 1 or 2 because the source data may be an odd byte which must be swapped. This is the only difference between D12.80 (destination modes 1 or 2) and D12.90. After one of these states or D12.00 has been completed, the processor performs a three-way branch, to separate JMP, JSR, and floating-point instructions, and instructions that transfer the source operand to the destination unchanged (specifically, the MOV, MTPI, and MTPD instructions) from all others. For floating-point instructions, the external data transfer is aborted, and the sequence continues through the B Fork decision point to FOP.40. For JMP instructions, the sequence is directed to JMP.00; for JSR instructions, to JSR.00. For the three direct-transfer (0 Class) instructions, the external transfer is forced to be a DATO instead of a DATIP or a DATI, and the transfer is completed before an instruction-dependent, condition-code load operation is performed. The last machine state in the sequence for 0 Class instructions also begins the instruction fetch for the next instruction and checks for asynchronous conditions requiring service.

For all other instructions, the DATI or DATIP transfer is completed, and the B Fork logic is conditionally enabled in D12.10. If a byte swap is needed because the destination address is to an odd byte, the extra machine state D12.30 is entered, and then the B Fork decision point. Note that in all three of the sequences shown (in D12.60, D12.10, and D12.70) the destination register is incremented by a constant which can be either 0, 1, or 2, depending on the address mode and whether a word or a byte operand is being fetched.

Destination Mode 3

The three sequences for destination mode 3 all enter D30.10 (destination mode 3), which completes the data transfer, increments the destination register by the necessary amount, and transfers to D10.20, which begins the fetch of the operand addressed by the word just transferred. Because the first transfer during a destination mode 3 sequence can only be a full word, the increment used in the register update is always 2, not 1.

1.2.5.6 FLOWS 6

Destination Modes 4 – 7

Flows 6 illustrates six machine state sequences that are used to fetch the destination operand when the destination address mode is 4, 5, 6, or 7. These six sequences correspond to the six sequences for address modes 1, 2, and 3.

Modes 4 and 5 require that the contents of the destination register be decremented before the value is used in the external data transfer. They are treated by one of three sequences. Modes 6 and 7 use general register 7 (the PC) first and then use the destination register. They are treated by one of three sequences.

In either case, two of the three sequences are entered from the C Fork and one from the A Fork. The two C Fork entries differentiate between source operands that require byte swapping and source operands that do not. There can be no requirement for a byte swap on the A Fork entry, because the source operand would be address mode 0 and the high byte of a register cannot be specified.

C Fork Entries for Modes 4 and 5
D45.80 (destination mode 4 or 5) and D45.90 differ mainly in the microprogram addresses contained in the microprogram word. Each state decrements the DR by the value of the destination constant, which is 1 for a byte operation in mode 4, and 2 for a word operation. Byte operations in mode 5 use a constant of 2 because the data fetched from the address taken from the DR is in turn used as an address and must be a full word. The state following D45.80 or D45.90 begins the external data transfer, which may be a DATI, DATIP, or a DATO, depending on the specific instruction. D40.30 and D50.30, which follow D45.90, also perform the byte-swap operation on the source operand. In each of the two sequences, a different path is taken for destination mode 4 where only one data transfer is needed, than for destination mode 5 where a second transfer is needed. The second transfer is performed by a sequence that is common for address modes 3, 5, and 7; this sequence transfers the first word that is fetched from the BR to the DR and then uses the DR as the address for a second transfer.

A Fork Entry for Modes 4 and 5
D45.00, which is entered from the A Fork Decision point, is similar to D45.80 and D45.90, except that a BEND is performed to abort the transfer begun during the IRD.00 machine state. The sequences that follow D45.00 are similar to the sequences that follow D45.80 or D45.90, except that the source operand, if any, is already in the SR.

Destination Modes 6 and 7 Entry
For address modes 6 and 7, the first machine state entered from the C Fork decision point begins an external data transfer, using the contents of the PC as an address, and performs an increment operation on the PC. The entry from the A Fork decision point continues the transfer begun by the IRD.00 machine state, so this entry is to D67.00 (destination mode 6 or 7) that follows the first state for the other entries. D67.10 adds the contents of the DR to the data read into the BR, thus performing the indexing operation, and then transfers to a machine state in the flow sequence for destination modes 4 or 5. The transfer is to D10.30 (a state also used for mode 4) if the mode is 6, or to D10.10 (a state also used for mode 5) if the mode is 7. The shared sequences perform the remaining one or two data transfers to fetch or store the actual data word.

Ending Sequence
When the last data transfer has been started, all six sequences enter a combined conditional fork and two-way branch that selects the next machine state. For 0/class instructions (MOV, MTPI, and MTPD) the last data transfer is a DATO operation, which is completed by D10.40; this state also loads the condition codes. The processor then returns to the instruction fetch sequence. For all other instructions, the DATI or DATIP transfer is completed in D10.60, leaving the destination data in the BR and the source data in the SR, and the B Fork logic is conditionally enabled. If a byte-swap operation is required for the destination data, D12.30, which performs this operation for all destination modes 1 - 7, is entered. FJ/Class instructions go directly to the B Fork.

1.2.5.7 FLOWS 7

ASH, ASHC, and Floating-Point Instructions
Flows 7 illustrates the machine state sequences for the Arithmetic Shift (ASH) and Arithmetic Shift Combined (ASHC) instructions, and the first machine state of the floating-point instruction service after the destination address calculation.

ASH Instruction
When the machine state sequence for the ASH instruction is entered from the B Fork decision point, the destination data is in the BR register. The six least-significant bits of the destination word are used as a 2's complement number which is the shift count for the instruction. The DR is loaded from the BR and this data is then loaded into the Shift Counter (SC) from the DR in ASH.10. In an ASH.20, the condition codes are loaded, based on the value of the word in the source register, and the

SC is tested for a 0 shift count. If the shift count is 0, the instruction is completed, and the processor returns to the instruction fetch sequence; otherwise, one of two states is entered, depending on the sign of the shift count. ASH.30 (Arithmetic Shift) and ASH.40 perform the actual shift one bit at a time, and increment or decrement, respectively, the shift counter. These states also load the condition codes with the results of each shift, so that after the last shift the codes are correct, and test during each cycle to determine whether any further cycles are required. Note that the first change to the SC is performed in ASH.20; all tests are done on the value before any changes are performed, so the last cycle in ASH.30 or ASH.40 is performed with the SC=0, and the final value in the SC is -0 (all 1s).

ASHC Instruction

The ASHC instruction operates in a manner similar to the ASH instruction. The difference is that two words of data are shifted. ASC.00 and ASC.10 perform the same functions as ASH.00 and ASH.10, and in addition, load the DR (after the SC has been loaded from the previous value in the DR) with the contents of a general register which is selected by ORing the destination register specification with 1. When the destination register specified by the instruction is an even-numbered register, the OR produces the number of the next higher numbered register.

ASC.20 performs the first change of the SC, moves the first data word to the BR, loads the condition codes, and tests for a 0 SC, just as ASH.20 does. However, if the SC is 0, the sequence continues with ASC.80 (arithmetic shift combined), instead of returning immediately to the instruction fetch sequence. This state is required to test the second data word, so that the Z condition code can be set on the contents of both words. ASC.80 also starts the next instruction fetch, so the processor transfers to either FET.10 or BRK.00 rather than FET.00.

If the SC is not 0, ASC.20 is followed by ASC.30 or ASC.40. These states perform the same operations as the corresponding states for the ASH instruction, and also cause shifting of the DR (which can be shifted internally, without passing the data through the ALU or SHFR). The bit shifted into the DR is selected by processor hardware. When the SC does reach 0, the next machine state is SC.60, which performs the same operations as ASC.80, but also stores the second word from the DR into the appropriate general register.

Floating-Point Instructions

When the B Fork logic decodes a floating-point instruction, FOP.40 (floating-point operation) is entered. This state aborts the last external data transfer started by the destination-data-fetch sequence, and sends the destination address, not the destination data, to the BR. A three-way branch is then entered:

1. BRQ true: Control is transferred to FSV.70 (Flows 13). In this cycle and the two that follow it, the original DR and PC are read back from the FP11-C and the DR, PCA and PCB are restored to the state in which they were prior to the FPP instruction fetch. The service flows (BRK.00 through SVC.90), and the interrupt subroutine are then executed; the FPP instruction is then fetched and executed again.

2. -(SYNC+BRQ): The processor cycles on FSV.60 (Flows 13) until it receives either an FP SYNC or a BRQ. In this last case it executes the sequence described in (1) above. In the first case (FP SYNC) it executes the sequence in (3) below.

3. SYNC+-BRQ: FSV.10 is entered. In this state, a bus cycle is started, whose direction (DATI or DATO) is determined by FC (BC←FC).

   a. If the instruction is not a Floating Pause Class (FPCLASS), up to four 16-bit words are transferred by the FSV.10–FSV.70 loop.

   b. If the instruction is FPCLASS, this loop is expanded to include FSV.30, FSV.40 and FSV.50 which cause the loop to execute a read/modify/write operation. FPCLASS instructions are ABSX and NEGX.

After the CPU completes this loop, it executes FSV.20 where it can copy the floating condition codes in the FP11-C, if desired. From this state, the CPU sequences to FET.07 to start the next instruction fetch.

1.2.5.8 FLOWS 8

Multiply Instruction
The sequence of machine states shown on Flows 8 performs a multiplication operation on two words of data, one from a general register and the other in a word specified by the destination field and fetched into the BR. The results of the multiplication are stored in two general registers: one is the register specified in the instruction, and the other is a register whose number is formed by ORing 1 with the number of the specified register (Figure 1-7). If the specified register has an odd number, only one register is used.

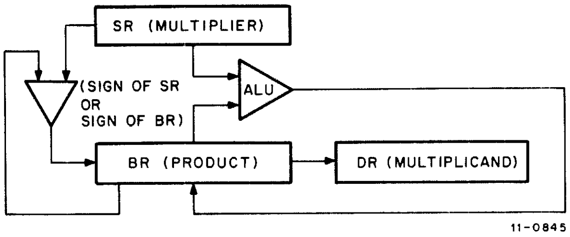


Figure 1-7 Multiply Instruction

The multiplier is in the SR, the multiplicand in the DR, and the 32-bit product is formed in the BR and DR by an add and shift algorithm.

The multiplier (SR) is used as a 32-bit, not a 16-bit, 2's complement number. This is accomplished by extending its sign bit into the BR after every shift. The multiplication thus has as its operands a 16-bit multiplicand, the DR, and a 32-bit multiplier, the SR.

In 2's complement notation, a negative 16-bit number (-A) is equivalent to (2^{16} -A), and a negative 32-bit number (-B) to (2^{32} -B). When a combination of 16- and 32-bit positive and negative numbers are multiplied, four conditions are possible, as shown in Table 1-2.

Note that correction of the product is required when the DR (multiplicand) is negative.

In Case 1, where both SR and DR are positive, the product is correct and no correction is required.

In Case 2, 2^{32} X DR must be subtracted, but since the product is only 32 bits wide, this term is out of range and no correction is required.

In Case 3, 2^{16} X SR has to be subtracted from the product, as this term is within the 32-bit product formed in the BR and DR.

In Case 4, the first two terms are out of range, and 2^{16} X SR must be added to the product. Since in this case the SR is a 2's complement negative number, the addition is accomplished by subtracting it as in Case 3 (- - = +).

The multiplication sequence begins with two machine states that set up the four registers (BR, SR, DR, and SC) used in the sequence, and performs the first test and shift on the DR. Note that all branches refer to the state of the DR and the SC at the beginning of the machine state preceding the branch, not the values in the registers at the end of that state. This is because the RAR is clocked at T3. The operand supplied by the destination-data-fetch sequence is loaded into the DR, and the SC is loaded with the octal value 17 (decimal 15) in MUL.00 (multiply).

In MUL.10, the BR is cleared; the other operand is in the SR as the result of IRD.00. The SC is decremented.

Fifteen multiplication cycles are then performed in MUL.20 and MUL.30.

1. **If the low order bit of the DR is 1 [DR0(1)], the SR is added to the BR and both BR and DR are shifted right in a combined shift, which forms the product (MUL.20).**

Table 1-2
Sign Correction for MUL Instruction

Case	SR	DR	Representation of		Product Generated (2^n SR X DR)	Product Should Be:	Correction Required
			SR	DR			
1	≥0	≥0	SR	DR	(SRXDR)	(SRXDR)	None
2	<0	≥0	2^{32}-SR	DR	2^{32}DR-(SRXDR)	-(SRXDR)	None
3	≥0	<0	SR	2^{16}-DR	2^{16}SR-(SRXDR)	-(SRXDR)	-2^{16}SR
4	<0	<0	2^{32}-SR	2^{16}-DR	2^{48}-2^{32}DR-2^{16}SR+(SRXDR)	(SRXDR)	+2^{16}SR

2. If the low order bit of the DR is 0 [DR0(0)], the shift is performed, but no add (MUL.30).

At the end of these fifteen cycles, SC=0 and DR0 contains the sign bit of the multiplicand (DR).

1. If DR0(1), the multiplicand was negative and correction is required. MUL.50 subtracts the multiplier (SR) from the high order product (BR and DR). This is the same as subtracting $2^{16} \times$ SR from the product.

2. If DR0(0). no correction is required (MUL.40).

MUL.50 or MUL.40 store the more-significant half of the result into the register specified by the source field, and set the condition codes on the value of this word.

MUL.60 stores the less-significant half of the result in the register, whose number is formed by ORing the source field with 1; if an odd register is specified, this value replaces the more-significant half of the result, which is lost. This is done because many multiplications produce a result which can be contained in only one word, and this result is preserved by this action. The condition codes are altered to represent the value of the entire result; if all 32 bits are 0, the Z bit is set, and if the result cannot be contained in one word, the C bit is set. At the end of this cycle, the sequence returns either to the instruction fetch sequence, or, if an asynchronous condition needing service was sensed by the BRQ STROBE in machine state MUL.40 or MUL.50, to the break service sequence.

1.2.5.9 FLOWS 9 and 10

The Divide Instruction

Division is the process of counting the number of times one number (the dividend) can be reduced by another number (the divisor). The count of the number of reductions is called the quotient; the part of the dividend that cannot be reduced by the divisor is called the remainder. Division is more complicated than multiplication, for several reasons:

1. Division produces two results, not one.

2. During multiplication, the maximum result occurs when the maximum number is multiplied by itself. This result fits into two words; during division, the maximum result occurs when the largest possible number is divided by a very small number and the result does not fit into any reasonable number of words. Therefore, the division algorithm must recognize the overflow condition when the quotient is too large.

3. During the division process, it is necessary to recognize when the partial remainder is smaller than the divisor; usually this is done by recognizing when the last reduction passed through 0 and changed the sign of the remainder. This condition is called underflow and requires that the results of the last reduction be restored in some way.

The simplest division algorithm is to subtract the divisor from the dividend until underflow occurs, restore the remainder, and keep a count of all but the last subtraction for the quotient (this algorithm assumes all positive numbers). This procedure is very tedious, particularly if an overflow condition exists, so a shorter algorithm is used that is based on the positional representation of numbers.

The result of the division is a quotient that can be multiplied by the divisor to regenerate the dividend (with a difference equal to the remainder). If, during the multiplication, each bit of the quotient can generate a partial product that becomes part of the total sum, then during the division, each bit of the quotient can be generated individually while reducing the partial remainder by an appropriate amount. To determine what the most-significant bit of the quotient should be, the number that is subtracted from the dividend is equal to the divisor, multiplied by the positional value of the most-significant digit.

Figure 1-8 illustrates the division algorithm. At the beginning of the division, the dividend occupies all of a word register. The divisor has been multiplied by 2 to the nth power, so that the number which is first subtracted from the dividend is actually the divisor times the positional value of the most-significant bit. Before each step of the division, the divisor is divided by 2, so that the correct number for generating the next bit of the quotient is formed; the division by 2 is done by shifting the 2-word divisor 1 bit to the right. In order for the division algorithm to operate with negative numbers, the reduction that is performed at each step of the division must be the correct operation to reduce the remainder; if the divisor and the partial remainder

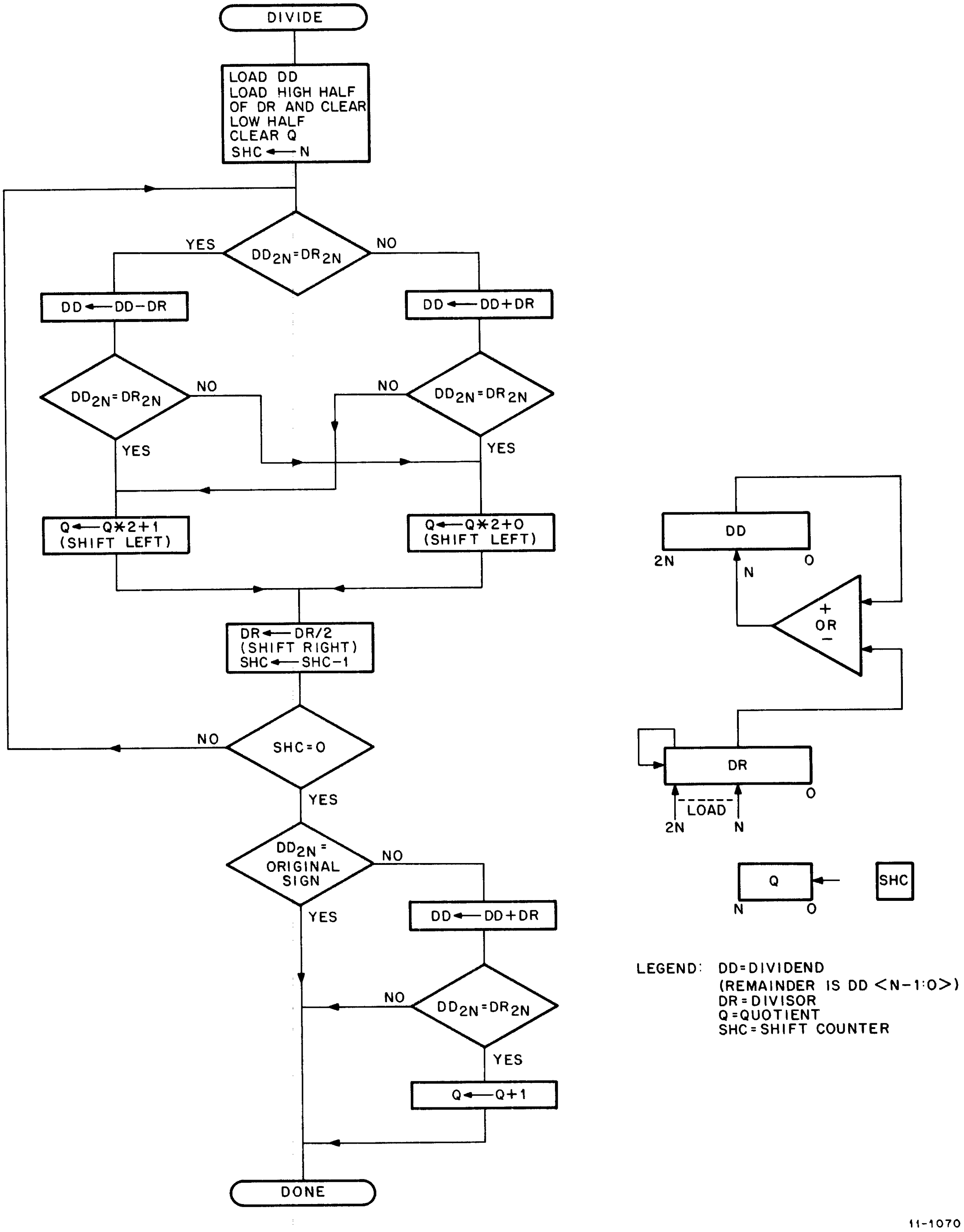


Figure 1-8 Divide Algorithm

(that is, the dividend) have the same sign, the divisor is subtracted from the remainder, but if their signs differ, the divisor is added to the remainder to reduce its magnitude.

The algorithm that is illustrated does not perform a restoration if an underflow condition occurs. Instead, while underflow exists, succeeding operations are performed in the opposite manner to complete the restoration; while an underflow condition exists, the bits of the quotient are set only when the underflow is corrected and are cleared if the operation does not complete the restoration. If the original divisor and dividend are of opposite sign, the quotient should be negative, so bits of the quotient depend on the operation performed and its results, as follows:

1. If the operation was a subtraction (the signs of the divisor and the partial remainder were the same), the quotient bit is set if there was no underflow, and is cleared if there was underflow.

2. If the operation was an addition (the signs of the divisor and the partial remainder were different), the quotient bit is cleared if there was no underflow, and is set if there was underflow.

The non-restoring division algorithm works because an underflow at any step can be corrected to within one multiple of the divisor by the succeeding steps. This is true because a binary number that is represented by all 1s is changed to a number that is represented by a 1, followed by all 0s, when the number 1 is added to it. Therefore, the multiple of the divisor that is subtracted from the partial remainder at any step is only one more multiple of the divisor than can be expressed by all the less-significant bits of the quotient. The remaining single multiple of the divisor can be restored by a single operation (which is always an addition, because underflow exists and the divisor and partial remainder have different signs) following the steps that generate the quotient bits; this step is also used to correct the remainder.

Divide Instruction Sequence
The divide (DIV) instruction is executed by the longest and most complex sequence of machine states used in the KB11-C Processor. This sequence is illustrated on two drawings. Flows 9 shows the register setup, the first two overflow tests, and the cycle of states that perform the actual division. Flows 10 shows the quotient and remainder sign corrections and the final overflow test.

The division is performed by a non-restoring divide algorithm that is described above. The hardware implementation (Figure 1-9) uses the SR to hold the divisor and begins with the dividend in the BR and DR registers. The BR contains the more-significant half of the dividend, while the less-significant half is in the DR. Each cycle of the division shifts the dividend one bit to the left and shifts the next bit of the quotient into the least-significant bit of the DR. When the division terminates, the quotient is in the DR and the remainder is in the BR.

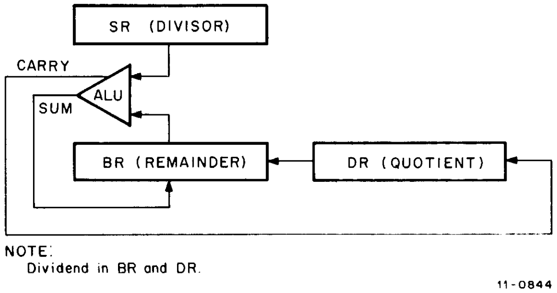


Figure 1-9 Divide Instructions

The non-restoring divide algorithm can operate with positive or negative operands; however, the KB11-C always operates on a positive dividend to simplify the detection of underflow. (The divisor may have either sign.) The first two machine states of the division sequence test for a 0 divisor or a negative dividend, and set up the SR and DR registers. If a 0 divisor is sensed, the division is aborted and the C, V, and Z condition codes are set to indicate that an error has occurred.

Initial Setup
If the dividend is negative, a sequence is entered to complement the dividend. Note that the branch on the N condition code occurs after DIV.20, although the condition code is loaded in DIV.10 (divide), because the branch condition must be available at the beginning of the machine state in which the branch is used. Similarly, the branch on the Z condition code after DIV.10 uses the condition code value set by DIV.00, not the new value set by DIV.10.

Negative Dividend Processing

The sequence beginning with DVN.00 (divide negation) generates the 2's complement of the 2-word dividend as follows:

1. The 2's complement of the less-significant word is formed by first clearing the DR, then subtracting the SR, which contains the low order word, from the 0 in the DR. The DR is cleared so that a subtract from 0, which requires only one machine state, can be used; normally a 2's complement is generated by forming the 1's complement and then incrementing, as shown for the remainder of correction steps. The 2's complement of the less-significant word is stored in the register which originally held the less-significant word.

2. DVN.20 generates a carry from the less-significant word to the more-significant word. That is, if a carry-out of the most-significant bit of the ALU occurs during the operations (which is repeated in DVN.20), a 1 is shifted into the DR.

3. A 1 is subtracted from the DR. If a carry occurred in Step 2, the DR contains 0 and the 2's complement of the more-significant word is formed; if no carry occurred, the DR now contains a -1, which cancels the carry insert during the subtraction in DVN.40, and the 1's complement of the SR is formed. This is the correct result if there is no carry.

After the 2's complement of the dividend is formed, DVN.50 begins the restoration of the divisor to the SR and the dividend to the BR and DR. However, if the dividend is still negative, which occurs if the dividend was the maximum negative number (because the 2s complement notation can express one more negative number than positive number, the largest negative number complements to itself), the division cannot be performed and the sequence is aborted.

Overflow Test and First Cycle

After the setup is completed, the processor enters DIV.30 with a positive dividend in the BR and DR, 17(8) in the SC, and the divisor in the SR. The next portion of the sequence performs the first cycle of the division and performs a test for overflow. This test is based on the fact that if underflow does not occur during the first cycle, the quotient is too large to be expressed in 16 bits. If the instruction is not aborted because of overflow, the processor enters the DIV.70 machine state to begin the main divide cycle.

Division Process

The test for underflow that determines whether DIV.80 or DIV.90 is entered is based on the following considerations:

1. If the divisor is negative, adding the divisor to the dividend should produce a result closer to 0 than the original dividend. If the result is negative, underflow has occurred and a 0 is shifted into the DR.

2. If the divisor is negative and the dividend is also negative, an underflow condition already exists. The divisor is subtracted from the dividend to return the dividend to a positive number. If the result is still negative, a 0 is shifted into the DR; if the result is positive, the underflow has been corrected and a 1 is shifted in.

3. For a positive divisor and dividend, a subtraction is performed. If the result is positive, a 1 is shifted into the DR, but if the result is negative, underflow has occurred and a 0 is shifted in.

4. If the divisor is positive and the dividend is negative, an addition is performed to correct an existing underflow. If the result is positive, the underflow has been corrected and a 1 is shifted into the DR, otherwise a 0 is shifted in.

As a result of these considerations, the processor enters DIV.80 if the divisor is positive and there is no underflow (DR0 is a 1), or if the divisor is negative and there is underflow (DR0 is a 0). DIV.80 performs a subtract operation and shifts the carry-out of the ALU into the DR. (A carry-out of the most-significant bit of the ALU indicates that underflow has occurred; if an uncorrected underflow existed, the carry indicates that it has been corrected.)

If the opposite conditions exist (SR is positive and DR0 is 0, or SR is negative and SR0 is 1), DIV.90 is entered and an addition is performed, followed by a shift of the DR. Note that the cases for which a carry-out of the most-significant bit of the ALU exist are equivalent to the cases described above for which the least-significant bit of the DR is set.

Remainder Storage and Sign Check

After the divide cycle has been performed 15 times (the first division cycle) and the first decrement of the SC is performed in DIV.30 - DIV.60, DVC.00 (divide correction) writes the remainder from the BR into the appropriate general register, and transfers control to one of four machine states, depending on whether a remainder correction is required and whether the quotient has the correct sign.

Remainder Correction

If, after the last division cycle, the least-significant bit of the quotient is a 0, an underflow condition still exists. This condition can be corrected (unless an overflow condition also exists) by adding a positive divisor or subtracting a negative divisor to correct the remainder. This is done by DVC.10 or DVC.20. If no remainder correction is needed, or following the remainder correction, DVC.30 or DVC.40 begins complementing the remainder in case the remainder has the wrong sign. The current value of the remainder is not disturbed until a determination is made of the appropriate sign.

Quotient Sign Change

If the N condition code is set, the original dividend was negative. The complemented remainder, which is negative because the corrected remainder is positive (if all underflow conditions are corrected), is stored as the final value of the remainder. If both the dividend and the divisor were positive, the quotient, which is also positive (the most-significant bit of the quotient must be positive or an immediate overflow condition aborts the division), is written into the appropriate general register. Similarly, if both dividend and divisor are negative, the quotient should be positive and is written in its present form.

If the original signs of the dividend and divisor were different, the quotient should be negative. The quotient is complemented by DVC.80 and DVC.90; one special case in which the quotient is the most negative number is considered an error.

1.2.5.10 FLOWS 11

Memory Reference Execution Sequences

Flows 11 illustrates eight sequences that execute the data manipulation stages of a variety of instructions, when those instructions require external data transfers to complete the instruction execution. These sequences are entered from the B Fork decision point.

Standard Execution

The majority of instructions are executed by EXC.00 (execute). When this state is entered, the source operand, if any, is in the SR, and the destination operand is in the DR. EXC.00 performs one data manipulation operation and loads the condition codes; both the operation performed and the condition-code loading are controlled by subsidiary ROMs (i.e., they are instruction-dependent). EXC.00 performs the byte-swap operation in the SHFR automatically.

For any instruction that is operating on an odd-byte destinaton operand, EXC.00 also begins an external data transfer operation that is completed in EXC.10; this operation transfers the result data to the destination address, which is taken from the DR.

Negate Instructions

Several instructions, which are otherwise treated in the same manner as those executed by EXC.00, must be executed separately. The negate and negate byte (NEG.B) instructions require two machine states for execution because the 2's complement of a number is formed by first generating the 1's complement and then incrementing that value. After the negation is performed and the condition codes loaded, the processor performs a byte swap if the destination operand is an odd byte, and starts an external data transfer that is completed in EXC.10.

Shifter Instructions

Two instructions, which are executed by EXC.00 when they operate on an even byte [DR0(0)], use the SHFR to perform a right shift. These are the ASRB and ROR instructions. When these instructions operate on a destination operand taken from an odd-byte location [DR0(1)], a second machine state is required to perform the byte swap, which also requires the SHFR. Therefore, SHR.00 (shift right) performs the same actions as EXC.00, except that no external data transfer is begun and no byte swap is performed. These functions are performed by SHR.10. No conflict occurs for the ASL and ROL instructions because left shifts are performed by the ALU, not by the SHFR.

Test Instructions

The three instructions that set the condition codes without modifying any stored data, TST, CMP, and BIT, are executed by machine states that do not start an external data transfer for the data operand.

Jump Instruction

The jump (JMP) instruction performs only one operation; it sets a new value in the Program Counter (PC). The value loaded into the PC is the destination address, not the destination data word. The last external data transfer to fetch the data word is aborted, (BEND) the PC is loaded, and a transfer to the instruction fetch sequence is performed by the machine state JMP.00 (jump).

Jump to Subroutine Instruction

The jump to subroutine (JSR) instruction performs two data transfers in addition to loading the PC. The contents of a register specified by the instruction are saved on the hardware stack, and the previous value in the PC is saved in the specified register. JSR.00 (jump to subroutine) the last external data transfer, loads the destination address into the PCA (but does not load the PCB from the PCA, so that the PCB can be stored in the general register until JSR.40), and loads the SR with the contents of the specified register. JSR.10 transfers the SR to the BR, which is the register that holds data to be transmitted during external data transfers, and loads the DR with the contents of general register 6, the Stack Pointer (SP). JSR.20 decrements the SP by 2 (to allocate a word at the top of the stack for the data to be stored); the new value is stored in the SP and in the DR for use in the external data transfer started in JSR.30. JSR.40 transfers the contents of the PCB to the specified general register and loads the PCB from the PCA. The data transfer begun in JSR.30 is completed in this state.

Move From Previous Space Instructions

The MFPI or MFPD instruction transfers data from the destination address to the hardware stack; it acts like a "push" instruction. If Memory Management is on, the address space from which the desination data is taken may differ from the address space that the data is pushed into, but this does not affect the operations within the processor.. The MFP.00 state is entered with the data to be transferred in the BR; this state loads the condition codes and loads the SR from the hardware stack pointer. The MFP.80 machine state is entered if the destination mode is 0; this implies that the data is in a general register. This data is loaded into the DR while the bus operation started by the IRD.00 machine state is aborted. The MFP.90 machine state transfers the DR to the BR and loads the SR from the stack pointer. The sequence for destination mode 0 then joins the sequence for the other address modes in MFP.10. This state decrements the SR (which contains the SP). SVC.80 and SVC.90 (Flows 13) complete the instruction by pushing the data onto the stack.

1.2.5.11 FLOWS 12 and 13 – Flows 12 and 13 show the abort, trap, interrupt and floating-point service routines. The abort, trap and interrupt sequences are described in Chapter 6. The FP11-C instructions are described in Paragraph 1.2.5.7 (Flows 7).

1.2.5.12 FLOWS 14 – Flows 14 shows the sequences for manual Console operations. These operations are described in Part III of this manual (Console).

1.2.6 Following an Instruction Through the Flowcharts

To follow a particular instruction through the flowcharts, it is necessary to know which machine state sequences apply to that instruction in the particular state of the processor (specifically, which machine state will be entered from various fork decision points).

The tables and diagrams in this paragraph are designed to help determine the exact sequence of machine states for a particular instruction. Starting with either the binary code, or the symbolic name of the instruction, the machine state entered from each decision point, and what branches are taken at some of the primary branch points within the sequences shown can be determined.

1.2.6.1 Figures and Tables – Figure 1-10 shows the correspondence between binary op codes and instruction mnemonics.

1. Starting with the most-significant bit of the instruction code, look down the corresponding column of Figure 1-10 to find the number that matches the value of that bit in the instruction.
2. The horizontal line to the right of that number leads to another vertical column, for the next most-significant group of bits in the binary code. Look down that line to find the number that matches the value of the corresponding bit or bits in the instruction.
3. Repeat Step 2 for each portion of the binary code until the last number is followed by the symbolic name and structure of an instruction instead of a horizontal line. That instruction corresponds to the given binary code.

When the symbolic code for an instruction is known, the reader can find that instruction in Table 1-3 which specifies the machine state sequences used to execute that instruction. The table is in alphabetical order according to the mnemonic codes used for the instructions, and lists both the instruction classes, if any, and the machine states entered from various decision points, when used. The instruction classes are groupings of the instructions according to properties of the execution sequences (e.g., I, P, and O/Class instructions perform a DATI, DATIP, or DATO bus transfer as the last transfer of the destination data fetch sequence). While the A Fork decision point is used by all instructions (the A Fork decision point follows the instruction fetch sequence and is, in effect, the instruction decoding system), not all instructions use the B Fork or C Fork decision points; those which do not are indicated by entry "N.U." in the appropriate column.

Table 1-3A
Instruction Microprogram Properties

Instruction	Class	A Fork	B Fork	C Fork
ADC.B	P, E, DAC	See Table 1-3B	EXC.00 (11)	N. U.
ADD: -SM0	P, E, BIN	See Table 1-3B	EXC.00 (11)	See Table 1-3D
SM0	P, E, BIN, DAC	See Table 1-3C	EXC.00 (11)	N. U.
ASH -DM0	DAC	See Table 1-3C	ASH.00 (7)	N. U.
DM0	DAC	ASH.10 (3)	ASH.00 (7)	N. U.
ASHC -DM0	DAC	See Table 1-3C	ASC.00 (7)	N. U.
DM0	DAC	ASC.10 (3)	ASC.00 (7)	N. U.
ASL.B	P, E, DAC	See Table 1-3C	EXC.00 (11)	N. U.
ASR	P, E, DAC	See Table 1-3C	EXC.00 (11)	N. U.
ASRB DR0 (0)	P, E, DAC	See Table 1-3C	EXC.00 (11)	N. U.
DR0 (1)	P, E, DAC	See Table 1-3C	SHR.00 (11)	N. U.
Branch Instructions: BCC (BHIS), BCS (BLO), BEQ, BGE, BGT, BHI, BHIS – See Table 1-3E				
BIC.B -SM0	P, E, BIN	See Table 1-3B	EXC.00 (11)	See Table 1-3D
SM0	P, E, BIN, DAC	See Table 1-3C	EXC.00 (11)	N. U.
BIS.B -SM0	P, E, BIN	See Table 1-3B	EXC.00 (11)	See Table 1-3D
SM0	P, E, BIN, DAC	See Table 1-3C	EXC.00 (11)	N. U.
BIT.B -SM0	I, E, BIN	See Table 1-3B	TST.10 (11)	See Table 1-3D
SM0	I, E, BIN, DAC	See Table 1-3C	TST.10 (11)	N. U.
Branch Instructions: BLE, BLO, BLOS, BLT, BMI, BNE, BPL – See Table 1-3E				
BPT (OP3)	None	TRP.00 (3)	N. U.	N. U.
Branch Instructions: BR, BVC, BVS – See Table 1-3E				
CCOP	None	CCP.00 (3)	N. U.	N. U.
CLR.B	P, E, DAC	See Table 1-3C	EXC.00 (11)	N. U.
CMP.B -SM0	I, E, BIN	See Table 1-3B	TST.10 (11)	See Table 1-3D
SM0	I, E, BIN, DAC	See Table 1-3C	TST.10 (11)	N. U.
COM.B	P, E, DAC	See Table 1-3C	EXC.00 (11)	N. U.
DEC.B	P, E, DAC	See Table 1-3C	EXC.00 (11)	N. U.
DIV -DM0	I, DAC	See Table 1-3C	DIV.00 (9)	N. U.
DM0	I, DAC	DVS.00 (3)	DIV.00 (9)	N. U.
EMT	None	RSD.00 (3)	N. U.	N. U.
Floating Point:	F, FJ			
-FP PRESENT		RSD.00 (3)	N. U.	N. U.
FP PRES*-DM0		FOP.00 (2)	FOP.40 (7)	See Table 1-3D
FP PRES*DM0		FOP.00 (2)	FOP.40 (7)	FOP.50 (4)
HALT	None	HLT.00 (3)	N. U.	N. U.
INC.B	P, E, DAC	See Table 1-3C	EXC.00 (11)	N. U.
IOT	None	TRP.00 (3)	N. U.	N. U.
JMP -DM0	J, FJ, DAC	See Table 1-3C	JMP.00 (11)	N. U.
DM0		RSD.00 (3)	N. U.	N. U.
JSR -DM0	J, FJ, DAC	See Table 1-3C	JSR.00 (11)	N. U.
DM0		RSD.00 (3)	N. U.	N. U.
MARK	None	MRK.00 (2)	N. U.	N. U.
MFP -DM0	I, DAC	See Table 1-3C	MFP.00 (11)	N. U.
DM0	I, DAC	MFP.80 (3)	N. U.	N. U.
MOV -SM0	O, E, BIN	See Table 1-3B	N. U.	See Table 1-3D
SM0	O, E, BIN, DAC	See Table 1-3C	N. U.	N. U.
MOVB -SM0	P, BIN	See Table 1-3B	EXC.00 (11)	See Table 1-3D
SM0	P, BIN, DAC	See Table 1-3C	EXC.00 (11)	N. U.
MTP	O	MTP.00 (1)	N. U.	See Table 1-3D
MUL -DM0	I, DAC	See Table 1-3C	MUL.00 (8)	N. U.
DM0	I, DAC	MUL.80 (3)	MUL.00 (8)	N. U.
NEG.B -DM0	P, DAC	See Table 1-3C	NEG.00 (11)	N. U.
DM0	P, DAC	NEG.70 (3)	N. U.	N. U.
RESET	None	RES.00 (3)	N. U.	N. U.
ROL.B	P, E, DAC	See Table 1-3C	EXC.00 (11)	N. U.
ROR	P, E, DAC	See Table 1-3C	EXC.00 (11)	N. U.
RORB DR0 (0)	P, E, DAC	See Table 1-3C	EXC.00 (11)	N. U.
DR0 (1)	P, E, DAC	See Table 1-3C	SHR.00 (11)	N. U.
RTI	None	RTI.00 (2)	N. U.	N. U.
RTS	None	RTS.00 (2)	N. U.	N. U.
RTT	None	RTI.01 (2)	N. U.	N. U.
SBC.B	P, E, DAC	See Table 1-3C	EXC.00 (11)	N. U.
SOB	None	SOB.00 (2)	N. U.	N. U.
SPL	None	SPL.00 (3)	N. U.	N. U.
SUB -SM0	P, E, BIN	See Table 1-3B	EXC.00 (11)	See Table 1-3D
SM0	P, E, BIN, DAC	See Table 1-3C	EXC.00 (11)	N. U.
SWAB	P, E, DAC	See Table 1-3C	EXC.00 (11)	N. U.
SXT	P, E, DAC	See Table 1-3C	EXC.00 (11)	N. U.
TRAP	None	RSD.00 (3)	N. U.	N. U.
TST.B	I, E, DAC	See Table 1-3C	TST.10 (11)	N. U.
WAIT	None	WAT.00 (3)	N. U.	N. U.
XOR	P, E, DAC	See Table 1-3C	EXC.00 (11)	N. U.

Table 1-3B
A Fork, BIN*-SM0

SOURCE 1-3

Source Mode	Machine State
1	S13.00 (1)
2	S13.01 (1)
3	S13.01 (1)
4	S45.00 (1)
5	S45.00 (1)
6	S67.00 (2)
7	S67.00 (2)

page of the FLOW

Table 1-3C
A Fork, DAC

Destination Mode	Machine State
0	(DF7 + BRQ):EXC.90 (3), -(DF7 + BRQ):EXC.80 (3)
1	D12.00 (5)
2	D12.00 (5)
3	D30.00 (5)
4	D45.00 (6)
5	D45.01 (6)
6	D67.00 (6)
7	D67.01 (6)

Table 1-3D
C Fork, BIN

Destination Mode	SR0	Machine State
0	0	DF7:D07.10 (4), -DF7:D00.90 (4)
	1	DF7:D07.00 (4), -DF7:D00.80 (4)
1	0	D12.80 (5)
	1	D12.90 (5)
2	0	D12.80 (5)
	1	D12.90 (5)
3	0	D30.80 (5)
	1	D30.90 (5)
4	0	D45.80 (6)
	1	D45.90 (6)
5	0	D45.80 (6)
	1	D45.90 (6)
6	0	D67.80 (6)
	1	D67.90 (6)
7	0	D67.80 (6)
	1	D67.90 (6)

Table 1-3E
Branches
(All Cycles on Flows 1)

Instruction	Branch Successful		Branch Not Successful	
	BRQ Present	BRQ Not Present	BRQ Present	BRQ Not Present
BCC	BXX.03	BXX.00	FET.01	FET.11
BCS	BXX.04	BXX.01	FET.03	FET.13
BEQ	BXX.05	BXX.02	FET.03	FET.13
BGE	BXX.03	BXX.00	FET.02	FET.12
BGT	BXX.03	BXX.00	FET.02	FET.12
BHI	BXX.03	BXX.00	FET.01	FET.11
BHIS	BXX.03	BXX.00	FET.01	FET.11
BLE	BXX.05	BXX.02	FET.03	FET.13
BLO	BXX.04	BXX.01	FET.03	FET.13
BLOS	BXX.04	BXX.01	FET.03	FET.13
BLT	BXX.05	BXX.02	FET.03	FET.13
BMI	BXX.04	BXX.01	FET.03	FET.13
BNE	BXX.03	BXX.00	FET.02	FET.12
BPL	BXX.03	BXX.00	FET.01	FET.11
BR	BXX.05	BXX.02	(always successful)	
BVC	BXX.03	BXX.00	FET.01	FET.11
BVS	BXX.04	BXX.01	FET.03	FET.13

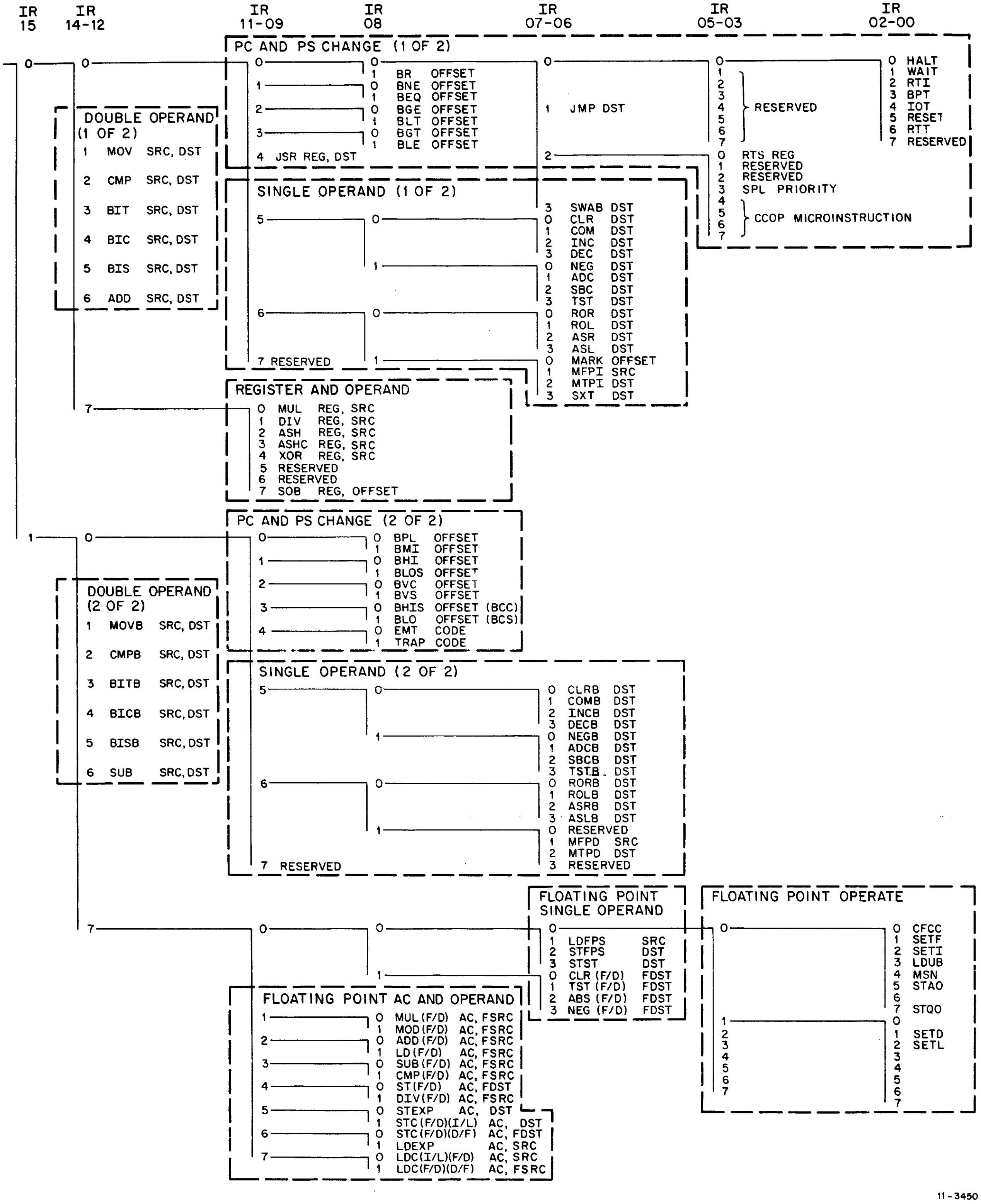


Figure 1-10 Determination of an Instruction from the Binary Code

Whenever possible, the entry for each active decision point specifies a machine state by its symbolic name, with the number of the flowchart where that state is illustrated in parentheses. If a particular machine state depends on additional conditions, those conditions are shown preceding the corresponding machine state and are separated from the state by a colon.

To follow an instruction through the Flows with Table 1-3, execute the following steps:

1. Find the instruction symbolic name in the INSTRUCTION column (Table 1-3A).

2. Go to the A Fork cycle shown under "A Fork" and follow the Flows until a B or C Fork, if any, is found.

3. Go to the B or C Fork cycle shown in Table 1-3A. Repeat Steps 2 and 3 if the instruction uses both the B and C Forks.

4. Determine the type of execute cycle from the CLASS column of Table 1-3.

A sample instruction is taken through the Flows, using this documentation, in Paragraph 1.2.6.2.

1.2.6.2 An Instruction Example – This paragraph traces one instruction through a sequence of machine states to illustrate the process of finding each machine state and using the flowchart and ROM map information to understand the operations performed by the processor. The example instruction and the environment in which it is executed is shown in Figure 1-11.

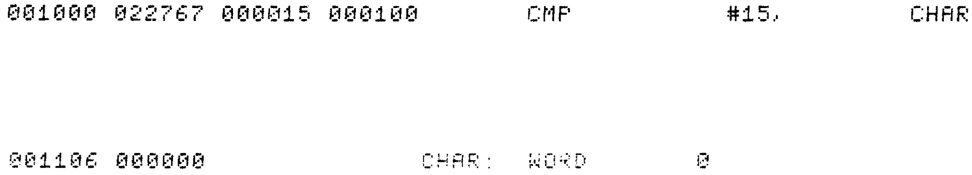
The instruction is a CMP, which subtracts the destination word from the source word and uses the result to set the condition codes. These may then be used by arithmetic and logic conditional branches. Its Source mode is 2 (SM2) and its source field (register) is 7 (SF7). After the Fetch cycles, the PC (register 7) contains 1002. This value is the address of the operand. A DATI is performed; it reads location 1002 which contains 15, the source operand.

The Destination mode is 6 (DM6) and the destination field is 7 (DF7). The PC contains 1006 after the source operand fetch. The destination operand is stored in the location whose address is the sum of the present PC (1006) plus the contents of the index word, whose address is 1004. The index word equals 100, and the destination operand is at location 1106. Two DATIs are required to obtain the destination operand: the first reads the index word, the second reads the operand.

Immediately before the processor begins the machine state sequence for this instruction, the Program Counter (PC) contains the value 1000(8), the processor status word contains the value 000340, there are no bus requests or other asynchronous conditions, and the processor is about to enter the FET.0X machine state. In this state, a DATI bus operation is begun, using the contents of the PC as the address.

FET.1X
Assuming that no requests have been strobed into the request register (refer to Chapter 6), the next machine state entered is FET.1X. In this state, the PC is updated (the new value is loaded into the PCA and does not disturb the PCB, which is still being used for the address in the data transfer) and the word that is read is loaded into the IR and BR. Thz PCA now contains 1002, the IR and BR contain 022767, and the PCB still contains 1000; finally, after the bus operation is completed, the PCB is updated to 1002.

IRD.00
The third machine state entered is IRD.00. In this state, the A Fork logic is enabled. According to Figure 1-10, the binary number in the IR represents a



```
001000 022767 000015 000100         CMP          #15.          CHAR



001106 000000                CHAR:  WORD         0
```

Figure 1-11 Instruction Execution Example

CMP instruction; the entry for this instruction in Table 1-3A refers to Table 1-3B, which indicates that for a source mode of 2 (as specified by the third octal digit of the instruction), the next machine state is S13.01. Since both the source and destination fields are 7, the IRD.00 machine state also loads the SR and DR with the updated PC value (1002). Since CMP is a binary instruction and its source mode is 2, RACH BUST is not asserted in IRD.00 (CONDITIONAL BUST, refer to Paragraph 1.2.5.1).

Source Operand

In S13.01 the DATI is started, using the contents of the SR as the address. The contents of the SR (1002) are incremented by 2, and this value is written back into the PCA and PCB, which now contain 1004.

The fifth machine state entered for this instruction is the S13.10 state. In this state, the DATI is completed, with the data that has been read-loaded into the BR register. The new'contents of the BR are 15 (the contents of the word following the instruction, which is the source operand). The DR is loaded with the updated contents of the register specified by the destination field of the instruction (because this is register 7, the DR is loaded from the PCB); the new contents of the DR is 1004.

Destination Operand

For a source mode of 2, the branch condition in S13.10 enables the Fork C logic. The entry for the CMP instruction in Table 1-3A refers to Table 1-3D, which indicates that, for a destination mode of 6 and the least significant bit of the SR equal to 0 [SR0(0)=even address], the next machine state is D67.80, which is shown on Flows 6. This machine state transfers the contents of the BR (=source operand) to the SR, and begins the third DATI bus operation, using the contents of the PCB as the address.

The next machine state is D67.00, which completes the third DATI and increments the PCA by 2. Because the DR is intended to reflect the current contents of the specified register, the DR is updated to reflect the new value in the PC, which is 1006. The data read into the BR is 100. This is the index word, which when added to the destination mode register (R7 or the PC), is the address of the destination word.

Following the D67.0 state, the processor enters the D67.1 state, where the PCB is loaded from the PCA and the contents of the BR is added to the contents of the DR. The result (1106) is the index word and is loaded into the DR. The branch condition in this machine state selects the D10.3 state to follow the D67.1 state (-DM357).

In the D10.3 machine state, the processor begins a fourth bus operation, using the contents of the DR (1106) as the address. The type of bus operation performed depends on the instruction class, according to Table 1-3A. A CMP instruction is an I/Class instruction, so a DATI operation is begun. This machine state also loads the BR from the SR, so that both registers contain 15.

The next state entered depends on the instruction class. A CMP instruction is not F, J, or O/Class, so the D10.60 state is entered. This state completes the fourth DATI operation, loading the contents of the location addressed by the DR (location 1106) into the BR. This word is the Destination Operand, which equals 0.

Execute

The D10.60 machine state branch condition enables the B Fork logic [DR0(0)]. The entry for a CMP instruction in Table 1-3A indicates that the next machine state is TST.10 (Flows 11).

The CMP instruction does not alter any data words, so no further bus operations are required. The TST.10 machine state performs instruction-dependent ($ on Flows) data operations and condition-code loading.

Flows 11 shows that the arithmetic operation is performed with the A operand = BR (destination word) and B = SR (source word). The ALU Control ROM Map on drawing GRAK shows that for CMP.B, the operation is A - B - 1, and that the SHFR does not change the result (except in the case of an odd byte operation, in which case the bytes are swapped).

In this example, the following operation is executed by the ALU:

A input:	0 000 000 000 000 000
B input:	-0 000 000 000 001 101
	1 111 111 111 110 011
minus 1:	-0 000 000 000 000 001
Result (to SHFR):	1 111 111 111 110 010 + carry

The condition codes are then set as shown by the CC Control ROM Map on drawing IRCJ:

N is set if "SHFR(15)0" (SHFR bit 15=0).

Z is set if "A=B(15:00)" (four-input gate to IRCF Z DATA1 L).

V is set if "A15*-B15*-ALU15+-A15*B15*ALU 15" (bottom two inputs to the lower IRCE VDATA L 74S65: A=AMX, B=BMX).

C is set if "ALU COUT 15" (DAPJ ALUCN L).

1. The N bit is cleared, since bit 15 of the SHFR is 1.
2. The Z bit is cleared, since the output of the ALU is not 0.
3. The V bit is cleared, since A15 and B15 (AMX bit 15 and BMX bit 15) are the same.
4. The C bit is set, since there is a carry from ALU bit 15.

NOTE

The arithmetic and the N and C condition code load operations are the opposite of those described in the First Edition of the PDP-11/70 Processor Handbook. The instruction, however, performs as specified in the Handbook.

1.3 ROM MAP

Refer to drawing D-CS-M8123-0-1, ROM & ROM CONTROL, sheets 12 – 15.

These four drawings list all the ROM states in numerical order. The following information is provided:

1. In the STATE column, the name by which the state is called on the Flows.
2. In the FLOWS column, the sheet of the Flow Diagrams on which the ROM state is shown.
3. In the ADR column, the ROM address of the state.
4. In the BRK – ALU columns, the value of each of the ROM fields for each state.
5. In the FEN column, the fork that is enabled, if any.
6. In the BEN column, the branch that is enabled, if any.
7. In the UAD column, the base address for the next ROM state, which may be modified if the FEN or BEN fields are other than 0.

1.4 ROM ADDRESS

Refer to Figure 1-12. The ROM Address Register (RAR), which is clocked at T3, determines the output of the ROM for the next cycle and supplies the address for the next cycle. It also supplies the address for the Memory Management ROM (refer to Section IV).

The input to the RAR [RACL RADR(07:00) H] is the address selection logic shown on RACL. The following are inputs to this logic:

1. The UADR field of the ROM. In the absence of any of the modifying signals, this is the ROM address for the next cycle.
2. The Branch inputs, which are controlled by conditions occurring in the rest of the processor logic.

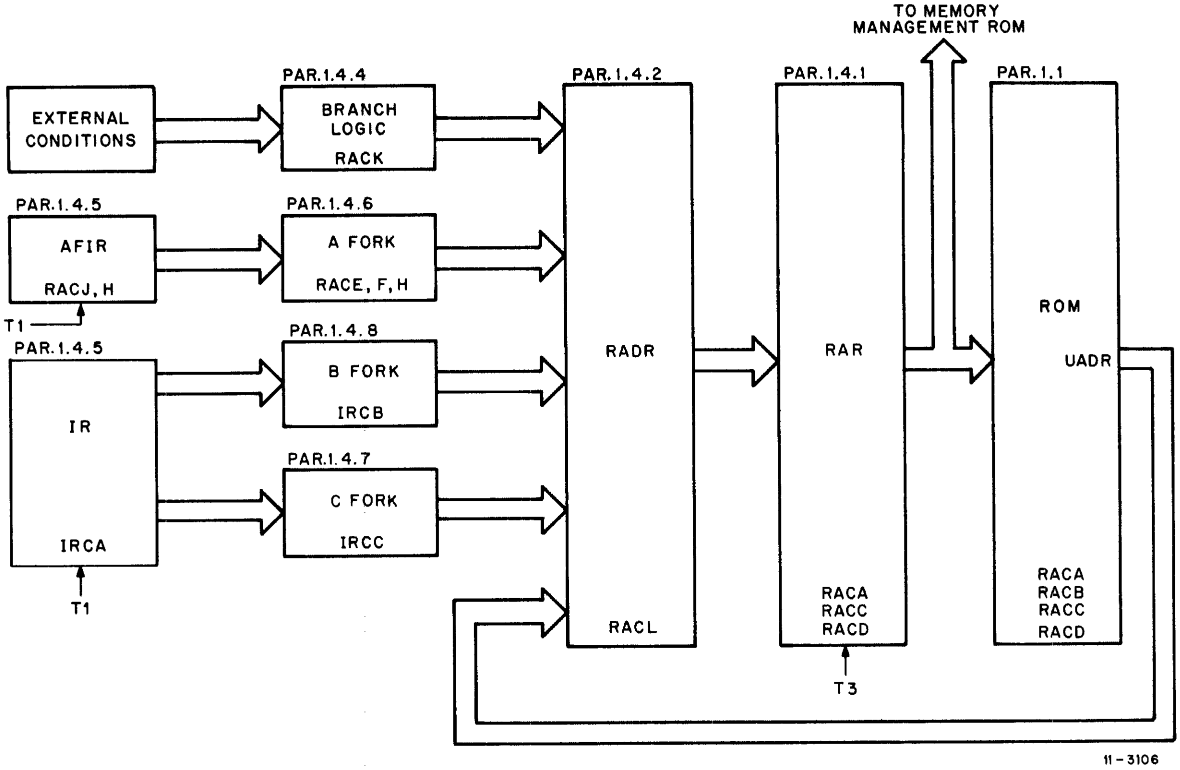


Figure 1-12 ROM Address

3. The Fork logic, which is controlled by the instruction word there are three Forks:

   a. The A Fork, used by all instructions, is the instruction decoder for the KB11-C.

   b. The C Fork, which is used only by binary instruction that require address calculation (SM not 0)

   c. The B Fork, which is used for execute cycles by instructions that require either source or destination address calculation, or both.

Figure 1-12 lists both the paragraph and the logic drawings containing information about the ROM address generation.

1.4.1 ROM Address Register (RAR)

There are three identical copies of the RAR. Refer to drawings RACA through RACD. In addition to the two copies (RARB and RARA) used to provide sufficient fanout for the 16 ROM ICs, a third copy (RAR, shown on RACD) is used to transmit the current microprogram word address to the Memory Management ROM (refer to Section IV of this manual).

The RAR is normally loaded from inputs generated by the microprogram address selection logic shown on drawing RACL. Under some circumstances, the RAR is forced to address 200 by clearing all but the most-significant of the eight bits, and setting that bit. To permit setting the most-significant bit, it is implemented by a separate flip-flop. The remaining seven bits are implemented by 6-bit registers of the same type used for the ROM output buffer.

RACA ZAP L is the signal used to force the processor into a known state to start the processing of aborts and of the power-up sequence. The conditions that can generate this signal are:

1. Power-up sequence or start sequence (ROM INIT)
2. Parity error abort, which is flagged UBCB PE ABORT during the microprogram cycle which follows a pause
3. All other aborts (TMCC ABORT), which are flagged during a pause cycle (RACB UBSD01).

PE ABORT and ABORT are gated with TIGD TS2 L, which remains asserted longer than the pulse TIGC T3 L that clocks the RAR, and ensures that the ZAP signal overrides the normal address.

ROM INIT and ABORT are described in Chapter 6 of this manual.

1.4.2 ROM Address Selection

Refer to drawing RACL. RADR(07:00) are the inputs to the RAR. An address bit is asserted (high), when all four of the negative-input-OR gates have at least one low input.

On all RADR 74S64 gates, there are four input OR gates. Three of these gates are used for the forks, one gate each for the A, B and C Forks. The fourth gate is the OR of the ROM UADR field bit and of the Branch Enable Bit (BRCAB) for that bit position. Since there is no branch enable for bit 3, the gate for RADR03 has only one input, UADR03.

1. When all three fork inputs are negated, the OR gate inputs for the forks are low. The inputs to the fourth gate then determine the state of the address bit: if either or both UADR and BRCAB bits are asserted (low), the RADR bit is asserted (high).
2. Only one of the three UFEN bits is ever asserted at one time (in a microprogram word). When one of these bits is asserted, its input to its RADR OR gates is high, and this OR gate is asserted if one or more of their fork logic input signals is asserted (low). In this case, the RADR bit is asserted (high).

From the above, it can be seen that:

1. A branch can assert an RADR bit for which the UADR is not asserted;
2. Any Fork can negate an RADR bit for which the UADR bit is asserted. For example, if UADR00 is asserted (low) and the A Fork (lower gate) is enabled, RADR00 is negated if none of the A0, A1, A2 RAB00 signals are asserted (low). The A Fork has an address of 377, or all eight UADR bits asserted; any combination of these could be negated to generate any address between 000 and 377.

Fork Inputs
The A Fork input, RACD UAFEN L is unconditional.

The C Fork input, RACD UCFEN L, is disabled by BEN14 if the source mode is 3, 5 or 7. This branch occurs during source mode operand fetch when one more bus cycle is required to fetch the source operand. Refer to Flows 1, S13.10 and Flows 2, S67.30: if -SM357, the next cycle starts the DM operand fetch on the C Fork; if SM357, both cycles fetch the operand in S13.20 - S13.40 and then go to the C Fork.

The B Fork input, RACD UBFEN L, is disabled by one of two conditions, both shown at the bottom of Flows 6:

1. BEN15. If the instruction is FJ/class, it goes directly to the B Fork for execution; if it is not FJ/class, it branches to one of two cycles, depending on whether or not it is O/class, to complete its destination operand fetch.
2. BEN05. An instruction that is neither O/class nor FJ/class goes to the B Fork if its destination address is not an odd byte. If it is an odd byte [DR0(1) or GRAB OBD] it first branches to D12.30 to swap bytes in the BR, and then goes to the B Fork.

1.4.3 Branches and Forks

Normally, the address of the next microprogram word is derived from the contents of the microaddress field (UADR) in bits 7 – 0 of the current microprogram word. Two Branch selectors allow 2-way or 4-way branches on the conditions of various processor circuits and on the contents of various data registers. For most decision points encountered during the flow of machine states, this branch capability is sufficient.

In certain situations, particularly after an instruction or data has been fetched by a state sequence that is common to many instructions, it is necessary to select a next machine state that is unique to one or a small class of instructions. This requires a much wider branching capability. In the **KB11-C** Processor, this capability is provided by the Fork logic. Each of three forks generates one of a large number of possible addresses, based on the decoding of the instruction, the address modes, and various processor status indications. When a fork is enabled by the corresponding fork-enable bit of the microprogram, the address generated by the fork is loaded into the ROM address register instead of the contents of the microaddress field.

1.4.4 Branch Logic

The processor is controlled by words fetched from a microprogram ROM; each word represents a machine state. The sequence of machine states is controlled by the sequence of ROM words fetched. Normally, each ROM word contains the address of the next word to be fetched. When it is necessary to provide for alterations in the sequence of machine states, two bits of the address contained in the current ROM word can be altered by inputs that sense processor conditions and data values. The altered bits select different addresses, depending on their final values, so that up to four different addresses can be selected. This 4-way branch permits a wide variety of machine state sequences to use the same microprogram words.

The two bits that can be altered by branch conditions are bits 5 and 4 of the microprogram address. Therefore, when a branch is used, the addresses selected for different conditions differ by 20, 40 or 60. There are 16 sets of branch conditions. One of the 16 sets is selected by the four branch-enable bits in the current microprogram word.

The Console branch (Flows 14) can modify bits 7, 6 and 2:0; it is not included in the explanation that follows, but is described in Section III (Console) of this manual.

RACK BRCAB(05:04) L are the outputs of the branch logic; each signal is ORed with the corresponding bit of the microprogram address from the current ROM word on one of the input gates to RACL RADR(05:04). When the 4-way branch is used, bits 5 and 4 of the UAD address are both negated (high), and the two branch signals select one of four addresses. If only a 2-way branch is desired, one of the UAD address bits is asserted (low), and the corresponding branch bit is ignored, because the result of the OR is always asserted.

Refer to drawing RACK. BRCAB05 L and BRCAB04 L are both generated by identical logic circuitry, which consists of two multiplexers and a 4-input AND-NOR gate. UBEF(03:00) controls the circuit.

UBEF03 selects the multiplexer: when this signal is not asserted, the top multiplexer is enabled and the lower one disabled. The opposite occurs when UBEF03 is asserted.

UBEF(01:00) selects which input to each half of the multiplexer IC is selected. Each IC has two outputs.

UBEF02 selects which of the two outputs of the multiplexer selected by UBEF(01:00) is gated through the BRCAB gate.

When UBEF(03:00) = 00, the D1 inputs to the top multiplexers are selected. Since these are both ground, BRCAB(05:04) are both negated (high), and the corresponding ROM address bits, RACL RADR(05:04) follow the UADR(05:04) inputs, i.e.: the address is not modified.The same is true for UBEF = 14, which is the Console branch.

Table 1-4 shows the inputs for each branch.

1.4.5 Instruction Registers

The instruction word is read from memory during FET.10. It is clocked into the Instruction Registers at T1 of IRD.00; this is shown as T6 of FET.10 on the Flows.

Table 1-4
Branch Signal Sources

UBEF Value	RACK BRCAB 05 L	RACK BRCAB 04 L	Comments
00	GROUND	GROUND	No Branch
01	IRCD DM357 H	GRAE SR EQ ONE L	
02	IRCF Z2 (1) H	TMCB (PWRF + INTR) L	
03	GRAJ SC = 0 L	GRAJ SC05 L	
04	GRAJ DIV SUB L	IRCH N (1) H	
05	GRAB OBD (0) H	GRAJ DIV QUIT L	BRCAB05: Disable B Fork if OBD, Flows 6
06	DAPA BR14 L	SSRA PS RESTORE (1) H	
07	RACK BE 75 H	RACK FP REQ H	
10	UBCC RIP + FP SYNC H	FRMB FP CLASS L	
11	GRAJ SC = 0 L	GRAD DR00 H	
12	TMCA CONF (1) H	TMCB BRQ TRUE L	
13	TMCB PF(0) * (SF + TF) H	TMCB PF(0) * (SF + -TF) H	Service Flows, Flows 12
14	GROUND	GROUND	Console Branch, Flows 14 Disable C Fork if SM357, Flows 1&2
15	IRCB FJ CLASS L	IRCC 0 CLASS L	Disable B Fork if F/J Class, Flows 6
16	GRAD DR00 H	GRAH SR15 H	
17	RACK RIP + FP SYNC L *TMCB BRQ * (T+ CONF) L	TMCB BRQ * (T+ CONF) L	

There are two copies of the Instruction Register (IR):

1. RACJ AFIR(15: 00) (1) H, which is used only by the A Fork logic for reasons of speed. For this same reason, there is an extra copy of bits 9 and 10 [RACH AFIR(10A:09A) (1) H].

2. IRCA IR(15:00) (1) H, which is used by the B and C Forks, the Condition Code logic and the rest of the **KB11-C** logic.

Both copies of the IR are clocked at T1 when the UIRK bit of the microprogram field is asserted in FET.10.

1.4.6 A Fork Logic

1.4.6.1 Decode Logic – Refer to drawing RACE. The logic illustrated on this drawing is part of the A Fork. This fork operates as the instruction decoder of the processor. Immediately after the instruction has been loaded into the Instruction Register (IR) the A Fork begins to generate an address. Because this address must be available within one machine cycle, the A Fork is designed to operate at maximum speed. Therefore, the amount of decoding is minimized; classes of instructions are recognized and the bits that differentiate members of the class are used directly as low-order bits of the generated address. This technique can be understood by examining the address utilization by the forks. As an example, consider the selection of addresses by the A Fork for the group of instructions ranging from HALT to RTT. The binary op codes for all these instructions are identical except for the three least-significant bits. When the A Fork decode logic recognizes that all but the three least-significant bits are 0, bit 3 of the ROM address is set, and the three least-significant bits of the op code become the three least-significant bits of the address.

1.4.6.2 Address Bit Generation – The logic shown on drawing RACE generates address bits for certain classes of instructions. These bits are then ORed with other signals that generate the same bits for other classes of instructions to generate the A Fork address. The address is then combined with the address from the microprogram in a bit-clear operation as shown on drawing RACL.

The signal names indicate the use of each logic circuit as follows:

1. The fork signals that are connected to the microaddress logic on drawing RACL have names that include RAB (for ROM Address Bit), followed by the number of the address bit to which the signal is connected.

2. In some cases, a signal is connected to more than one address bit because the same conditions generate both bits.

3. Many RAB signals are connected to the same address bit. They are distinguished by a letter that tells which fork generates the bit, and where more than one signal can be generated for the same fork. Thus, the signal RACE A0 RAB00 is one of several signals used by the A Fork logic to generate bit 0 of the address.

Branch instructions are described separately in Paragraph 1.4.6.4.

Table 1-5 shows the RAB bits asserted by each instruction on the A Fork.

1.4.6.3 Instructions Other Than Branch

RACE A0 RAB (02:00)
RACE A0 RAB00 L, RACE A0 RAB01 L, and RACE A0 RAB02 L are used to generate microprogram addresses 001 – 007. No other A Fork bits are enabled when these gates are enabled. The enabling conditions for all three signals are identical, except that each signal corresponds to a different bit of the Instruction Register. The IR bits passed through the AND-NOR gates are the destination-mode bits for instructions that require Destination Address Calculation (DAC), but no source address calculation. If the destination mode is 0, the destination data is in the Destination Register and no address calculation is required.

This group of microprogram words is used for the following groups of instructions:

1. All single-operand instructions (with op codes of 005XDD, 105XDD, 006XDD and 106XDD); this includes the instruction group from CLR to ASL (in both word and byte forms), the variable address-space moves, SXT, and XOR. These instructions are recognized by their op codes and generate the signal RACE RCLASS H.

2. The register and memory instruction group, which includes MUL, DIV, ASH, and ASHC. When one of these instructions is decoded, the signal RACE (MUL:ASHC+MFP) H is generated.

3. Any binary instruction with a source mode of 0. Because the source data is already in the Source Register, it is not necessary to do the source data fetch. These instructions generate the signal RACE BIN*SM0 H.

4. The three instructions JMP, JSR, or SWAB. These three instructions use the same address calculation as the single-operand instructions. The signal RACE JMP + JSR + SWAB H is generated.

The instructions that use A0 RAB(02:00) are listed below:

00 01 DD	JMP	07 0R SS	MUL
00 03 DD	SWAB	07 IR SS	DIV
00 4R DD	JSR	07 2R SS	ASH
00 50 DD	CLR	07 3R SS	ASHC
00 51 DD	COM	07 4R DD	XOR
00 52 DD	INC	10 50 DD	CLRB
00 53 DD	DEC	10 51 DD	COMB
00 54 DD	NEG	10 52 DD	INCB
00 55 DD	ADC	10 53 DD	DECB
00 56 DD	SBC	10 54 DD	NEGB
00 57 DD	TST	10 55 DD	ADCB
00 60 DD	ROR	10 56 DD	SBCB
00 61 DD	ROL	10 57 DD	TSTB
00 62 DD	ASR	10 60 DD	RORB
00 63 DD	ASL	10 61 DD	ROLB
00 65 SS	MFPI	10 62 DD	ASRB
00 67 DD	SXT	10 63 DD	ASLB

IF SM0:

01 SS DD	MOV	11 SS DD	MOVB
02 SS DD	CMP	12 SS DD	CMPB
03 SS DD	BIT	13 SS DD	BITB
04 SS DD	BIC	14 SS DD	BICB
05 SS DD	BIS	15 SS DD	BISB
06 SS DD	ADD	16 SS DD	SUB

RACE A0 RAB03

RACE A0 RAB03 L is generated for the following groups of instructions:

1. Branch instructions accompanied by a Bus Request (BRQ); these instructions generate A Fork addresses ranging from 330 - 336. Refer to Paragraph 1.4.6.4.

2. Op codes 000000 - 000007; these instructions range from HALT to RTT and use microprogram addresses 010 - 017 (017 is for op code 000007 and traps through location 4).

   The instructions in this group are:

00 00 00	HALT	00 00 04	IOT
00 00 01	WAIT	00 00 05	RESET
00 00 02	RTI	00 00 06	RTT
00 00 03	BPT	00 00 07	

3. E/class instructions, with the exception of the binary instructions that have both SM0*DM0, if these instructions have a DF7 or there is a BRQ to be serviced (DF7+BRQ). These instructions all go to address 030 because AFIR(05:03) are all 0s, which causes RACE RAB(02:00) to be negated. RACF A2 RAB03 asserts bit 3 for BIN*SM0*DM0*(DF7+BRQ).

   The instructions in this group are:

00 03 DD	SWAB*DM0	00 61 DD	ROL
00 50 DD	CLR	00 62 DD	ASR
00 51 DD	COM	00 63 DD	ASL
00 52 DD	INC	00 67 DD	SXT
00 53 DD	DEC	07 4R DD	XOR
00 55 DD	ADC	10 50 DD	CLRB
00 56 DD	SBC	10 51 DD	COMB
00 57 DD	TST	10 52 DD	INCB
00 60 DD	ROR	10 53 DD	DECB

Table 1-5A
A Fork Address Generation

Instruction	Class	A0 RAB						A1 RAB					A2 RAB			Address & Flows
		00	01	02	03	05	07	00	01	02	04	05	00	03	05	
ADC.B	P, E, DAC	See Table 1-5B														
ADD: -SM0	P, E, BIN	See Table 1-5B														
SM0	P, E, BIN, DAC	See Table 1-5C														
ASH -DM0	DAC	See Table 1-5C														
DM0	DAC					X			X							052 (3)
ASHC -DM0	DAC	See Table 1-5C														
DM0	DAC					X		X	X							053 (3)
ASL.B	P, E, DAC	See Table 1-5C														
ASR	P, E, DAC	See Table 1-5C														
ASRB DR0 (0)	P, E, DAC	See Table 1-5C														
DR0 (1)	P, E, DAC	See Table 1-5C														
Branch Instructions: BCC (BHIS), BCS (BLO), BEQ, BGE, BGT, BHI, BHIS – See Table 1-7																
BIC.B -SM0	P, E, BIN	See Table 1-5B														
SM0	P, E, BIN, DAC	See Table 1-5C														
BIS.B -SM0	P, E, BIN	See Table 1-5B														
SM0	P, E, BIN, DAC	See Table 1-5C														
BIT.B -SM0	I, E, BIN	See Table 1-5B														
SM0	I, E, BIN, DAC	See Table 1-5C														
Branch Instructions: BLE, BLO, BLOS, BLT, BMI, BNE, BPL – See Table 1-7																
BPT (OP3)	None				X			X	X							013 (3)
Branch Instructions: BR, BVC, BVS – See Table 1-7																
CCOP	None									X		X				044 (3)
CLR.B	P, E, DAC	See Table 1-5C														
CMP.B -SM0	I, E, BIN	See Table 1-5B														
SM0	I, E, BIN, DAC	See Table 1-5C														
COM.B	P, E, DAC	See Table 1-5C														
DEC.B	P, E, DAC	See Table 1-5C														
DIV -DM0	I, DAC	See Table 1-5C														
DM0	I, DAC					X		X								051 (3)
EMT	None															000 (3)
Floating Point:	F, FJ															
-FP PRESENT																000 (12)
FP PRES													X			101 (2)
HALT	None				X											010 (3)
INC.B	P, E, DAC	See Table 1-5C														
IOT	None				X					X						014 (3)

Instruction	Class	A0 RAB						A1 RAB					A2 RAB			Address & Flows
		00	01	02	03	05	07	00	01	02	04	05	00	03	05	
JMP -DM0	J, FJ, DAC	See Table 1-5C														
DM0																000 (3)
JSR -DM0	J, FJ, DAC	See Table 1-5C														
DM0																000 (3)
MARK	None							X	X	X					X	047 (2)
MFP -DM0	I, DAC	See Table 1-5C														
DM0	I, DAC								X	X					X	046 (3)
MOV -SM0	O, E, BIN	See Table 1-5B														
SM0	O, E, BIN, DAC	See Table 1-5C														
MOVB -SM0	P, BIN	See Table 1-5B														
SM0	P, BIN, DAC	See Table 1-5C														
MTP	O							X		X					X	045 (1)
MUL -DM0	I, DAC	See Table 1-5C														
DM0	I, DAC					X										050 (3)
NEG.B -DM0	P, DAC	See Table 1-5C														
DM0	P, DAC						X						X			301 (3)
RESET	None				X			X		X						015 (3)
ROL.B	P, E, DAC	See Table 1-5C														
ROR	P, E, DAC	See Table 1-5C														
RORB DR0 (0)	P, E, DAC	See Table 1-5C														
DR0 (1)	P, E, DAC	See Table 1-5C														
RTI	None				X				X							012 (2)
RTS	None											X				040 (2)
RTT	None				X				X	X						016 (2)
SBC.B	P, E, DAC	See Table 1-5C														
SOB	None					X		X	X	X						057 (2)
SPL	None							X	X			X				043 (3)
SUB -SM0	P, E, BIN	See Table 1-5B														
SM0	P, E, BIN, DAC	See Table 1-5C														
SWAB	P, E, DAC	See Table 1-5C														
SXT	P, E, DAC	See Table 1-5C														
TRAP	None															000 (3)
TST.B	I, E, DAC	See Table 1-5C														
WAIT	None				X			X								011 (3)
XOR	P, E, DAC	See Table 1-5C														

Table 1-5B
A Fork, BIN*-SM0

Source Mode	A0 RAB						A1 RAB					A2 RAB			Address & Flows
	00	01	02	03	05	07	00	01	02	04	05	00	03	05	
1							X			X					021 (1)
2								X		X					022 (1)
3								X		X					022 (1)
4									X	X					024 (1)
5									X	X					024 (1)
6								X	X	X					026 (2)
7								X	X	X					026 (2)

Table 1-5C
A Fork, DAC

Destination Mode	A0 RAB						A1 RAB					A2 RAB			Address & Flows
	00	01	02	03	05	07	00	01	02	04	05	00	03	05	
0: -(DF7+BRQ)										X					020 (3)
0: BIN*(DF7+BRQ)										X			X		030 (3)
0: -BIN*(DF7+BRQ)				X						X					030 (3)
1	X														001 (5)
2		X													002 (5)
3	X	X													003 (5)
4			X												004 (6)
5	X		X												005 (6)
6		X	X												006 (6)
7	X	X	X												007 (6)

10 55 DD	ADCB	10 61 DD	ROLB
10 56 DD	SBCB	10 62 DD	ASRB
10 57 DD	TSTB	10 63 DD	ASLB
10 60 DD	RORB		

RACE A0 RAB04

RACE A0 RAB04 L is generated for any branch instruction. This signal is an input to bits 4, 6 and 7 of the microprogram address; as a result, all branch instructions generate A Fork addresses with these three bits set (addresses between 320 and 336). Refer to Paragraph 1.4.6.4.

RACE A0 RAB05

RACE A0 RAB05 L is generated for MUL, DIV, ASH, and ASHC instructions with a destination mode of 0, and for SOB instructions. RACE BIN L eliminates the binary instructions from U/class. This RAB signal is also connected to RAB03 to generate addresses ranging from 050 to 057.

These instructions are listed below:

07 0R SS	MUL	07 3R SS	ASHC
07 1R SS	DIV	07 7R NN	SOB
07 2R SS	ASH		

RACH A0 RAB07

RACH A0 RAB07 is asserted for a NEG or NEGB instruction with DM0. Together with RACH A2 RAB00, it generates address 301.

RACF A1 RAB(02:00)

RACF A1 RAB00 L, RACF A1 RAB01 L, and RACF A1 RAB02 L generate the three least-significant bits of the ROM address for the classes of instructions described in the following paragraphs.

HALT Through Op Code 7 – These instructions generate microprogram addresses ranging from 010 – 017; the 1 in bit 3 of the address is generated by RACE A0 RAB03 L. The following instructions are included in this group:

00-00 00	HALT	00 00 04	IOT
00 00 01	WAIT	00 00 05	RESET
00 00 02	RTI	00 00 06	RTT
00 00 03	BPT		

X/Class – The X/Class instructions, MARK, MFP with a destination mode of 0, and MTP, generate addresses of 074, 046, and 045, respectively. RAB02 is forced to a 1, and the two low-order bits are the complements of the corresponding bits from the Instruction Register. Bit 5 of the address is set by RACF A2 RAB05 L.

U/Class – U/Class instructions include three groups: the binary instructions; the SOB instruction; and the MUL, DIV, ASH, and ASHC instructions with a destination mode of 0.

The Binary instruction use four microprogram addresses, 021 for SM1, 022 for SM23, 024 for SM45, and 026 for SM67. These bits are controlled by AFIR(11:09); bit 0 (A1 RAB00) can only be set by SM1 [RACH BIN*(-SM01) L]. Bit 4 of these addresses is set by RACH A1 RAB04 [(-BF1=7)*(-BF1=0)*(-SM0) = op codes with bits 14:12 from 1 – 6 and not source mode 0]. The instructions in this group are:

01 SS DD	MOV	11 SS DD	MOVB
02 SS DD	CMP	12 SS DD	CMPB
03 SS DD	BIT	13 SS DD	BITB
04 SS DD	BIC	14 SS DD	BICB
05 SS DD	BIS	15 SS DD	BISB
06 SS DD	ADD	16 SS DD	SUB

MUL, DIV, ASH and ASHC with DM0 and SOB use addresses 050 – 053 and 57. Bits 11:09 of the op code generate bits 02:00 of the address; bits 3 and 5 of the address is asserted by RACE A0 RAB05.

RTS:CCOP – Op codes 0002XX (RST:CCOP) use addresses 040 – 044. Bit 0 of the address is set when IR(05:03) = 3 (SPL), bit 1 when IR(05:03) = 2 or 3 (OP22, Flows 3 and SPL), bit 2 when IR(05:03) = 4 (CCOP). Bit 5 of the address is set by RACF A1 RAB05. The instructions in this group include:

00 02 0R	RTS
00 02 10	Unused
through	
00 02 27	Unused
00 02 3N	SPL
00 02 40	NOP
00 02 41	CCOP
through	
00 02 77	CCOP

RACH A1 RAB04

RACH A1 RAB04 L is asserted for the following instructions:

1. Binary instructions with:

   a. Both source and destination modes 0 (addresses 20 and 30);

   b. Any source mode except 0 (addresses 21, 22, 24, and 26);

   The instructions in this group are the following, when either SM0*DM0 or SM(1:7).

01 SS DD	MOV	11 SS DD	MOVB
02 SS DD	CMP	12 SS DD	CMPB
03 SS DD	BIT	13 SS DD	BITB
04 SS DD	BIC	14 SS DD	BICB
05 SS DD	BIS	15 SS DD	BISB
06 SS DD	ADD	16 SS DD	SUB

2. R/Class instructions with destination mode 0, except MFP and the NEG.B instructions (addresses 20 or 30);

   The instructions in this group are the following, when DM0:

00 50 DD	CLR	07 4R DD	XOR
00 51 DD	COM	10 50 DD	CLRB
00 52 DD	INC	10 51 DD	COMB
00 53 DD	DEC	10 52 DD	INCB
00 55 DD	ADC	10 53 DD	DECB
00 56 DD	SBC	10 55 DD	ADCB
00 57 DD	TST	10 56 DD	SBCB
00 60 DD	ROR	10 57 DD	TSTB
00 61 DD	ROL	10 60 DD	RORB
00 62 DD	ASR	10 61 DD	ROLB
00 63 DD	ASL	10 62 DD	ASRB
00 67 DD	SXT	10 63 DD	ASLB

3. SWAB instructions with a destination mode of 0 (also addresses 20 or 30).

RACF A1 RAB05

RACF A1 RAB05 is asserted for RTS:CCOP except when IR(05:03) = 1 which are unused op codes.

RACH A2 RAB00

RACH A2 RAB00 generates bit 0 and 6 of the ROM address. It is asserted in the following cases:

1. For NEG.B instructions with DM0, address 301. RACH A0 RAB07 asserts bit 7 in this case.

2. For branch instructions when RACF TRUE1 is asserted. Refer to Paragraph 1.4.6.4.

3. For floating point instructions, address 101.

RACH A2 RAB(02:01)

These bits are used by the branch instructions. Refer to Paragraph 1.4.6.4.

RACF A2 RAB03

RACF A2 RAB03 asserts bit 3 of the address for E/class binary instructions (= both source and destination modes equal to 0; no address calculation), either when the destination field is 7 or a BRQ is to be serviced. RACE A0 RAB03 asserts bit 3 for the non-binary E/class instructions.

The instructions in this group are the following, when SM0*DM0 and (DF7+BRQ):

01 SS DD	MOV	11 SS DD	MOVB
02 SS DD	CMP	12 SS DD	CMPB
03 SS DD	BIT	13 SS DD	BITB
04 SS DD	BIC	14 SS DD	BICB
05 SS DD	BIS	15 SS DD	BISB
06 SS DD	ADD	16 SS DD	SUB

RACF A2 RAB05

RACF A2 RAB05 asserts bit 5 of the ROM address for MFP instructions with DM0, and for MARK and MPT instructions.

1.4.6.4 Branch Instructions – Table 1-6 lists the Branch Instructions, their op codes and the conditions on which they branch.

With the exception of BR, which always branches, the branch instructions are grouped in pairs, each of which checks one condition (e.g.: BNE and BEQ check the Z bit). Bit 08 of the op code determines whether the instruction branches when the branch condition is true (1 or asserted) or false (0 or negated). For example: BNE branches if Z=0 and BEQ branches if Z=1.

RACF TRUE1 and TRUE2 are asserted when the branch condition is met. TRUE1 checks the result of branches that have a 1 in bit 15 of their op code; TRUE2 does the same for branches with a 0 in bit 15 of their op code. These two functions cannot both be asserted at one time.

Branch A Fork Address

Table 1-7 shows the generation of RACL RADR(07:00) for branch instructions.

Refer to Flows 1. Branch instructions (BXX) are shown on three separate branches:

Table 1-6
Branch Instructions

Instruction	Branch Condition	AFIR								RACF (See Note 1)	
		15	14	13	12	11	10	09	08	TRUE2	TRUE1
BR	Always	0	0	0	0	0	0	0	1	1	0
BNE	Z	0	0	0	0	0	0	1	0	X	0
BEQ	Z	0	0	0	0	0	0	1	1	X	0
BGE	N∀V	0	0	0	0	0	1	0	0	X	0
BLT	N∀V	0	0	0	0	0	1	0	1	X	0
BGT	Zv (N∀V)	0	0	0	0	0	1	1	0	X	0
BLE	Zv (N∀V)	0	0	0	0	0	1	1	1	X	0
BPL	N	1	0	0	0	0	0	0	0	0	X
BMI	N	1	0	0	0	0	0	0	1	0	X
BHI	CvZ	1	0	0	0	0	0	1	0	0	X
BLOS	CvZ	1	0	0	0	0	0	1	1	0	X
BVC	V	1	0	0	0	0	1	0	0	0	X
BVS	V	1	0	0	0	0	1	0	1	0	X
BCC,BHIS	C	1	0	0	0	0	1	1	0	0	X
BCS,BLO	C	1	0	0	0	0	1	1	1	0	X

NOTE 1 – "X" in the RACF TRUE1 or TRUE2 columns means that the function is asserted if the "Branch Condition" is asserted. For example, if the instruction is a BNE or a BEQ, TRUE2 is asserted if the Z bit is set.

NOTE 2 – The op code (AFIR < 15:08) for each pair of Branch Instructions differs only in bit 08. If bit 08 is set, the instruction branches, if the Branch Condition is asserted. If bit 08 is not set, the instruction branches if the condition is not asserted. For example:

BNE	Z = 0	Branch
	Z = 1	No Branch
BEQ	Z = 0	No Branch
	Z = 1	Branch

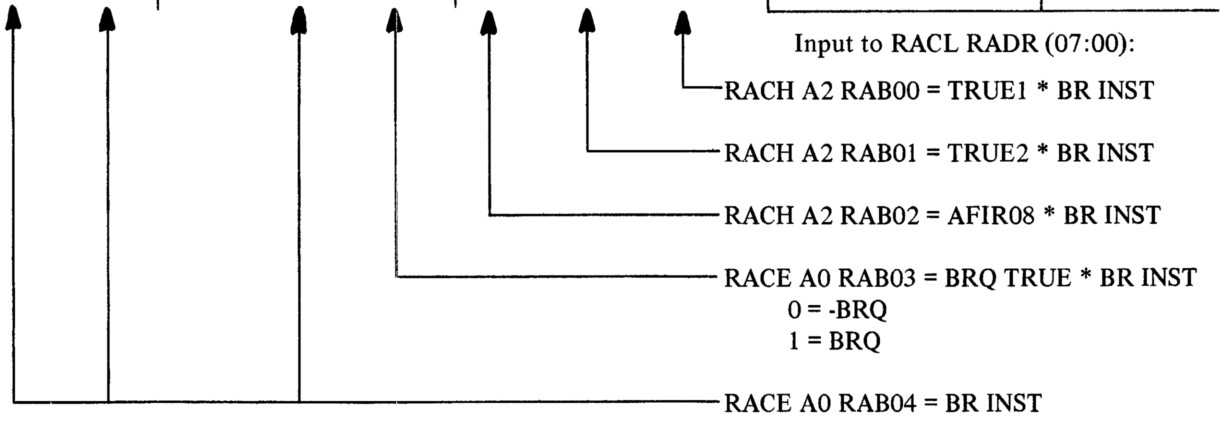
1. BXX*BCOK (Branch OK = condition met). In this case, cycles BXX.00 - BXX.05 (all identical) are executed. Since the branch is successful, the PC plus the displacement is moved to PCA and PCB, a BRQ strobe is issued, the bus cycle started in IRD.00 is ended, and the microprogram goes to FET.00. The instruction fetch sequence then fetches the instruction pointed to by the new PC.

2. BXX* - BCOK* - BRQ (condition not met and no break request). Since BRQ is not true and the instruction does not branch, control goes to FET.11 - FET.13.

3. BXX* - BCOK* - BRQ (condition not met and break request asserted). Control remains with the current PC, but the BRQ must be serviced; the next states are FET.01 - FET.03, after which the BRQ is serviced.

Table 1-7
Branch Instruction ROM Address

RACL RADR								Result	Next State
07	06	05	04	03	02	01	00		
1	1	0	1	0	0	0	0	BCOK * -BRQ	FET.0X
1	1	0	1	0	1	0	1		
1	1	0	1	0	1	1	0		
1	1	0	1	1	0	0	0	BCOK * BRQ	FET.0X
1	1	0	1	1	1	0	1		
1	1	0	1	1	1	1	0		
1	1	0	1	0	0	0	1	-BCOK * -BRQ	FET.1X
1	1	0	1	0	0	1	0		
1	1	0	1	0	1	0	0		
1	1	0	1	1	0	0	1	-BCOK * BRQ	FET.0X
1	1	0	1	1	0	1	0		
1	1	0	1	1	1	0	0		



Input to RACL RADR (07:00):

RACH A2 RAB00 = TRUE1 * BR INST

RACH A2 RAB01 = TRUE2 * BR INST

RACH A2 RAB02 = AFIR08 * BR INST

RACE A0 RAB03 = BRQ TRUE * BR INST
0 = -BRQ
1 = BRQ

RACE A0 RAB04 = BR INST

Refer to Table 1-7.

1. RACL RADR(07:06) H and RADR04 H are asserted (high) for all branch instructions by RACE A0 RAB04, which is a decode of all branch instruction op codes.

2. RACL RADR05 is negated (low) for all branch instructions.

3. RACL RADR03 is asserted (high) when BRQ is true during a branch instruction and negated when BRQ is not true. This bit is controlled by RACE A0 RAB03.

4. RACL RADR02 is asserted when bit 08 of the op code is 1 (branch if condition true).

5. RACL RADR01 is asserted by RACH A2 RAB01 when RACF TRUE2 is asserted.

6. RACL RADR00 is asserted by RACH A2 RAB00 when RACF TRUE1 is asserted.

It can be seen from Table 1-7 that a branch is successful (BCOK) under the following conditions:

1. When the instruction requires a branch on condition false or not asserted (RAB02 = 0) and neither TRUE2 nor TRUE1 are asserted (RAB01 = 0 and RAB00 = 0).

2. When the instruction requires a branch on condition true or asserted (RAB02 = 1) and either TRUE2 or TRUE1 are asserted (RAB01 = 1 or RAB00 = 1).

A branch is not successful (–BCOK) when the above conditions are not met, i.e.: RAB02 asserted and neither TRUE1 nor TRUE2 asserted, or RAB02 not asserted and either TRUE1 or TRUE2 asserted.

1.4.7 C Fork Logic

Refer to drawing IRCC. The logic shown on this drawing decodes the address modes and register specifications of the current instruction, and generates signals that control register selection and address calculation in the processor. The logic also generates addresses for the C Fork microprogram address logic. The C Fork selects the address of the next microprogram address when a destination operand must be fetched.

Two 8251-1 BCD-to-Decimal Decoders are used to recognize the source and destination modes, respectively, by decoding each 3-bit IR field. The source and destination modes determine the operations performed in the fetching of operands; these signals are used throughout the IRC module. Destination mode 0 is also used to separate the C Fork addresses for this mode and all other destination modes, by connecting IRCC DSTM0 L to the C Fork input for bit 7 of the ROM address (as shown on drawing RACL) and connecting IRCC DSTM0 H to the input for bit 6. In this manner, the C Fork generates microprogram addresses ranging from 202 – 211 for destination mode 0, and microprogram addresses ranging from 110 – 117 for other destination modes.

The address generated by the C Fork logic depends on:

1. For mode 0, whether or not the instruction is F/class. If it is not F/class, whether the destination field is 7 or not, and whether an odd byte swap is required (SR0 = 1 or 0);

2. For other modes, whether an odd byte swap is required.

The C Fork multiplexer is 74S157 4-bit 2-Line-to-1-Line Multiplexer that is controlled by IRCC DSTM0 L. Recognition of destination mode 0 generates the four low-order bits of the microprogram address for the C Fork. The two high-order bits are directly controlled by the destination mode and bits 4 and 5 are always 0. Bit 3 of the address is always a 1 if the destination mode is not 0 (the input is a ground which generates a low output, which asserts the input to the microprogram address assembly logic on drawing RACL). For destination mode 0, bit 3 is controlled by the instruction class; the bit is set for F/class instruc tions and clear for all others. Table 1-8 summarizes the C Fork multiplexer outputs.

Table 1-8
C Fork Address Generation

Instructions	Flows	ROM Cycle		C Fork Multiplexer				
		Adrs	Name	Input Enabled	Output: IRCC C0 RAB			
					03	02	01	00
DM0 * -F/Class * DF7 * SR0 (1)	4	202	D07.00	A	H	H	L	H
DM0 * -F/Class * DF7 * SR0 (0)	4	203	D07.10	A	H	H	L	L
DM0 * -F/Class * -DF7 * SR0 (1)	4	204	D00.80	A	H	L	H	H
DM0 * -F/Class * -DF7 * SR0 (0)	4	205	D00.90	A	H	L	H	L
DM0 * F/Class	4	211	FOP.50	A	L	H	H	L
DM12 * SR0 (1)	5	110	D12.90	B	L	H	H	H
DM12 * SR0 (0)	5	111	D12.80	B	L	H	H	L
DM3 * SR0 (1)	5	112	D30.90	B	L	H	L	H
DM3 * SR0 (0)	5	113	D30.80	B	L	H	L	L
DM45 * SR0 (1)	6	114	D45.90	B	L	L	H	H
DM45 * SR0 (0)	6	115	D45.80	B	L	L	H	L
DM67 * SR0 (1)	6	116	D67.90	B	L	L	L	H
DM67 * SR0 (0)	6	117	D67.80	B	L	L	L	L

1.4.8 B Fork Logic

Refer to drawing IRCB. The B Fork logic generates microprogram addresses that are used to select the next machine state after the destination operand has been fetched. For each instruction that operates on a destination operand, there is a unique microprogram word that controls the execution of the operation for that instruction. The majority of these instructions are included in the P/class group. The P/class instructions are executed by a single microprogram word that is stored in ROM location 031, with the exception of the NEG, ASRB, and RORB instructions. The exceptions are made because these instructions may require a byte swap during the execution cycle, and must use other machine states that permit a separate byte-swap operation for odd-byte data.

The B Fork addresses are generated by a 74S157 2-input, 4-bit multiplexer, and by two additional gates. IRCB B0 RAB04 L is connected to ROM address bits 4 and 5, to generate ROM addresses ranging from 60 – 67. IRCB B0 RAB03 L is connected to ROM address bits 3 and 4, to generate ROM addresses ranging from 31 – 36. The ROM addresses used by the B Fork and the instructions executed by each address, are listed in Table 1-9.

Table 1-9
B Fork Address Generation

Instructions	Flows	ROM Cycle		IRCB Multiplexer		Other Signals Asserted
		Adrs	Name	Inputs Enabled	Outputs Asserted	
P/Class * -[(ASRB + RORB) * DR0 (1) + NEGB]	11	031	EXC.00	A	B0 RAB00	B0 RAB03
TST.B + BIT.B + CMP.B	11	033	TST.10	A	B0 RAB01	B1 RAB00 B0 RAB03
JSR	11	034	JSR.00	A	B0 RAB02	B0 RAB03
JMP	11	035	JMP.00	A	B0 RAB02	B1 RAB00 B0 RAB03
F/Class	7	036	FOP.40	A	B0 RAB01 B0 RAB02	B0 RAB03
MUL	8	060	MUL.80	B	B0 RAB04	
DIV	9	061	DIV.00	B	B0 RAB00 B0 RAB04	
ASH	7	062	ASH.00	B	B0 RAB01 B0 RAB04	
ASHC	7	063	ASC.00	B	B0 RAB00 B0 RAB01 B0 RAB04	
[ASRB + RORB] * DR0 (1)	11	064	SHR.00	B	B0 RAB02 B0 RAB04	
MFP	11	066	MFP.00	B	B0 RAB01 B0 RAB02 B0 RAB04	
NEG	11	067	NEG.00	Multiplexer disabled, output all 1s.		

Note: All Signals on IRCB.

When the multiplexer is disabled for a NEG instruction, the outputs are all 1s: this generates address 67. For all other addresses, the inputs are selected by a signal that is generated for the MUL, DIV, ASH, ASHC, ASRB, RORB, and MFP instructions. When this signal is asserted, the B inputs of the multiplexer are used; RAB04 is forced to a logic 1 by a 0 V input. Conversely, the A inputs are used for F/class, J/class, K/class, and most P/class instructions; RAB04 is forced to a 0 by a +3 V input. The instructions that use the A inputs of the multiplexer also assert IRCB B0 RAB03 L. IRCB B0 RAB(02:00) L are generated by connecting the instruction group signals to the multiplexer inputs in the order required for each signal.

1.5 CONDITION CODES

The four least-significant bits of the PS word contain the processor condition codes. These bits store information about the value resulting from data manipulation during an instruction. The condition codes are not altered to reflect the results of address calculations, but are changed only when an instruction explicitly operates on a unit of data.

The condition codes can also be set to any specific value by transferring a word containing that value to the PS address. The value of the condition codes are altered by every interrupt or trap response function, and by every RTI or RTT instruction. In addition, individual condition-code bits may be manipulated directly, with the condition-code operate instructions. These instructions provide a means to set any one, or more, of the condition codes with a single instruction that requires only one memory reference; a similar set of instructions can clear any one or more bits. The condition codes are used in conditional branch instructions, so the various means of manipulating the condition codes are useful because they permit setting up the PS word to respond in a particular way to various branch instructions.

1.5.1 Condition Code Storage

Refer to drawing IRCH. The circuits shown on the top half of this drawing are used to store the processor condition codes; the remainder of the drawing shows circuits concerned with the subsidiary ROMs used in condition-code calculation, instruction decoding, and Arithmetic and Logic Unit (ALU) control.

The four condition-code bits, N, Z, V, and C, are stored in the four least-significant bits of the Processor Status (PS) word. The remaining bits of the PS, and the PS loading and reading logic, are on the PDR module and are shown on drawing PDRD. (Refer to Chapter 3, Control Registers.) The condition codes are normally loaded to reflect the result of each instruction that operates on data. When this is done (by clocking the data inputs to each flip-flop), each bit takes on the value of the corresponding signal from the condition code generation logic on drawings IRCE and IRCF. Two Z bit flip-flops, provided to avoid the delay of a final stage OR gate before the clock time, are shown on drawing IRCF.

Clocked Inputs - IRCH CCLK H clocks the condition-code flip-flops immediately following each ROM cycle (T6 is the T1 of the following cycle) except when the clock is inhibited by a value of 2 in the Condition Code Load (CCL) bits in the microprogram. In many cases where the condition codes are clocked, individual bits may remain unaffected by loading the bit from itself, through the combinational logic that generates the condition codes.

BR Inputs – The condition code flip-flops can be loaded directly from the BR. This is done whenever the bus address transmitted by the processor addresses the low byte of the Processor Status (PS) word. UBCB CC DATA (1) H indicates this condition and is used to gate the BR bits into the direct-set and direct-clear inputs of the flip-flops. Complements are applied to set and clear inputs, so that each flip-flop is correctly set or reset.

IR Inputs – A third method of modifying the condition codes allows bits to be set or cleared directly from the CCOP instruction. The four least-significant bits of the IR are connected to either the set or clear inputs of the flip-flops, but not both. The selection of inputs is done by two enabling signals that are generated from opposite polarities of IR04. The same polarity inputs from the IR are used for either setting or clearing; only bits which are 1s in the IR are altered, the remaining bits are not affected.

When the condition codes are set or cleared from the IR, the normal clocking of the flip-flops is inhibited. When the condition codes are loaded from the BR, the loading signal is present beyond the time when the data inputs are clocked, so the BR inputs take precedence. Unless one of these two conditions is true, the normal clocked input is used.

The Z bit is stored in two flip-flops shown on drawing IRCF. The flip-flop outputs are ORed to generate the value of the condition-code bit. If either flip-flop contains a 1, the Z bit is considered to be a 1. Both flip-flops are set or cleared together when either the BR or IR bits are transferred to the condition codes.

1.5.2 Condition Code Load Field

The Condition Code Load (CCL) field of the ROM is decoded as shown on drawing IRCF to determine how the PSW condition-code bits are to be altered. The CCL field is summarized in Table 1-10.

1.5.3 Instruction Dependent Control

When CCL = 1, the Condition Code loading is instruction dependent, i.e., controlled by the operation code field of the instruction; this control is implemented by two subsidiary ROMs, CC CNTL ROM and the INSTR DECODE ROM, both shown on IRCH.

1.5.4 SUBROM Address Generation

IRCH SUBROM(04:00) H is the address, for the Condition Code Control and Instruction Decode ROMs; it is also the address for the ALU Control ROM (refer to Chapter 2). This address is generated from IRCA IR(15:06), by the two multiplexers and the OR gate on drawing IRCH.

Each subsidiary ROM contains 32 8-bit words. The 32 addresses are organized as follows (addresses in octal):

a. Addresses 0–7 are used for instructions with op codes containing 06 in IR (14:09). These include the rotates, shifts, MARK, MFP, MTP, and SXT.

b. Addresses 10–17 are used for instructions with op codes containing 05 in IR (14:09). These are the single-operand instructions.

Table 1-10
Condition Code Load

RACA UCCL			Output Asserted IRCF:	Function
02	01	00		
0	0	0	CC NON AFF L	No change
0	0	1	CC INSDEP H	Instruction-dependent. Condition codes determined by subsidiary CC CNTL ROM.
0	1	0	(IRCH SETCC H)*	Set or clear CC; dependent upon IR.
0	1	1	CCFP LOAD L	Load CCs from floating-point processor
1	0	0	CCLD4	Z and N: ACC SHFR C and V: 0
1	0	1	CCLD5	Z and N: ACC SHFR C: AMX15 V: V old + (AMX ∀ ALU)
1	1	0	CCLD6	N, C, and V: not affected Z: Z* SHFR = 0
1	1	1	CCLD7	Z, N, and V: not affected C: carry

* Generated on drawing IRCH.

c. Addresses 20–27 are used for binary instructions [IR (14:12) contains any value from 1 to 6].

d. Addresses 30–37 are used for the register destination instructions, which have a 7 in IR(14:12). These include multiply and divide, the long shifts, and XOR.

Instructions included in a. and b. above, have subrom addresses equal to IR(09:06) via the D inputs to the multiplexers; SUBROMA4 is low.

For the register destination instructions, SUBROMA4 is asserted, SUBROMA3 is driven by a +3 V input to the multiplexer, and the remaining three address bits take on the value of IR (11:09) through the C inputs of the multiplexer. For binary instructions, the B inputs of the multiplexer are used; SUBROMA4 is asserted and SUBROMA3 is clear. This data is summarized in Table 1-11.

The SUB instruction is treated specially, to separate the ADD and SUB instructions when generating ROM addresses. Both SUB and ADD would normally generate ROM address 26 (the op codes differ only in bit 15). When the SUB instruction is decoded, the four least-significant bits of the ROM address are forced to 0s to generate address 20. Addresses 27, 35, and 36 are not used. For the SWAB instruction, which is not in any of the four groups that generate ROM addresses, the contents of the IR generate the same ROM address that is used for the ASL instruction. The signal IRCH SWAB L is used to distinguish between the two instructions. The UALU signals are used to recognize that the ALU control is instruction-dependent, and that the outputs of the ALU control ROM on drawing GRAA are active.

1.5.5 C Bit Data

The C (Carry) bit of the PSW is set when a processor operation causes a carry out of the most-significant bit. The logic that generates the C bit data is shown on drawing IRCF. Figure 1-13 is a simplified diagram of the logic that asserts IRCF CDATA L. Each AND gate input covers a group of instructions that could cause a carry. The notation adjacent to each AND gate indicates the conditions or instructions that enable the gate and the resultant C bit source that asserts IRCF CDATA L.

Table 1-12 lists the instruction-dependent CC CNTL ROM outputs that control the C bit for each group of instructions. IRCE WOB CARRY H and IRCE LOB CARRY H are derived from a 74S153 multiplexer. These C bit inputs are determined from AMX 00, AMX 07, or AMX 15.

1.5.6 N Bit Data

The N (negative) bit of the PSW is set when a negative result is produced by a processor operation. The logic that generates the N bit data is shown on drawing IRCF. Figure 1-14 is a simplified diagram of the logic that asserts IRCF NDATA L. Each AND gate input decodes a particular group of instructions or processor operations for which a negative result might be obtained.

For most of the instructions, the CC CNTL ROM outputs IRCH MODZN H and IRCH ENZN H are asserted. These control outputs condition the NDATA logic to examine the SHFR output to determine when the N bit should be set. For word or odd-byte operations, the input A logic tests SHFRA15, and sets N accordingly. For byte operations, the input C logic tests SHFRA07. These inputs control the N bit for most operations.

Table 1-11
Subsidiary ROM Address Sources

Type of Instruction	ROM Address Multiplexer Select		Input Selected	Subsidiary ROM Address Source				
	S1	S0		A4	A3	A2	A1	A0
IR(14:09) = 05 or 06	H	H	D	0	IR09	IR08	IR07	IR06
Register destination	H	L	C	1	1	IR11	IR10	IR09
Binary	L	H	B	1	0	IR14	IR13	IR12
Not used	L	L	A			Not Used		

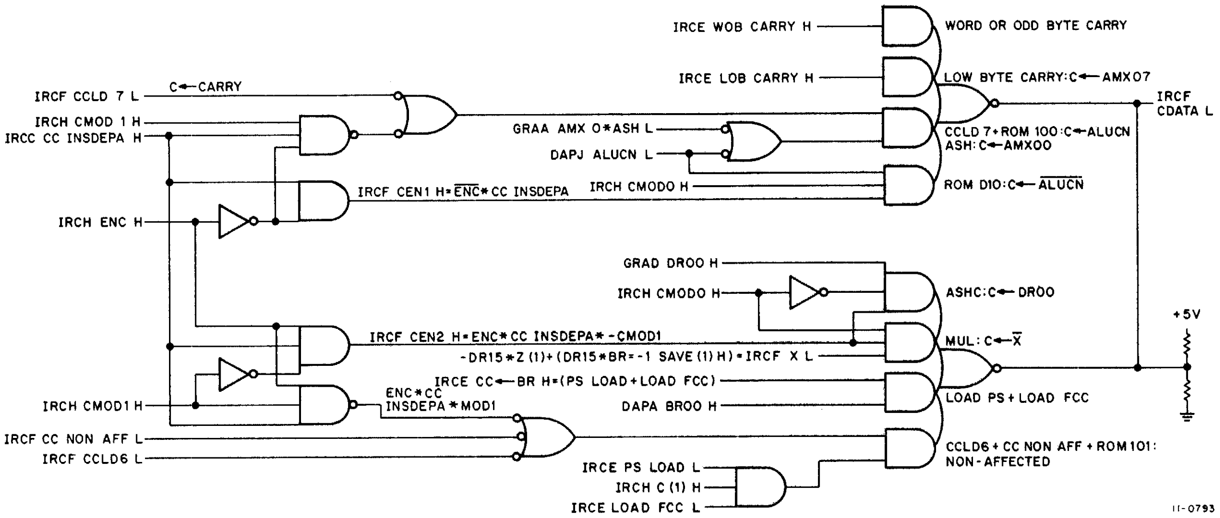


Figure 1-13 Sources of C Bit Data, Simplified Diagram

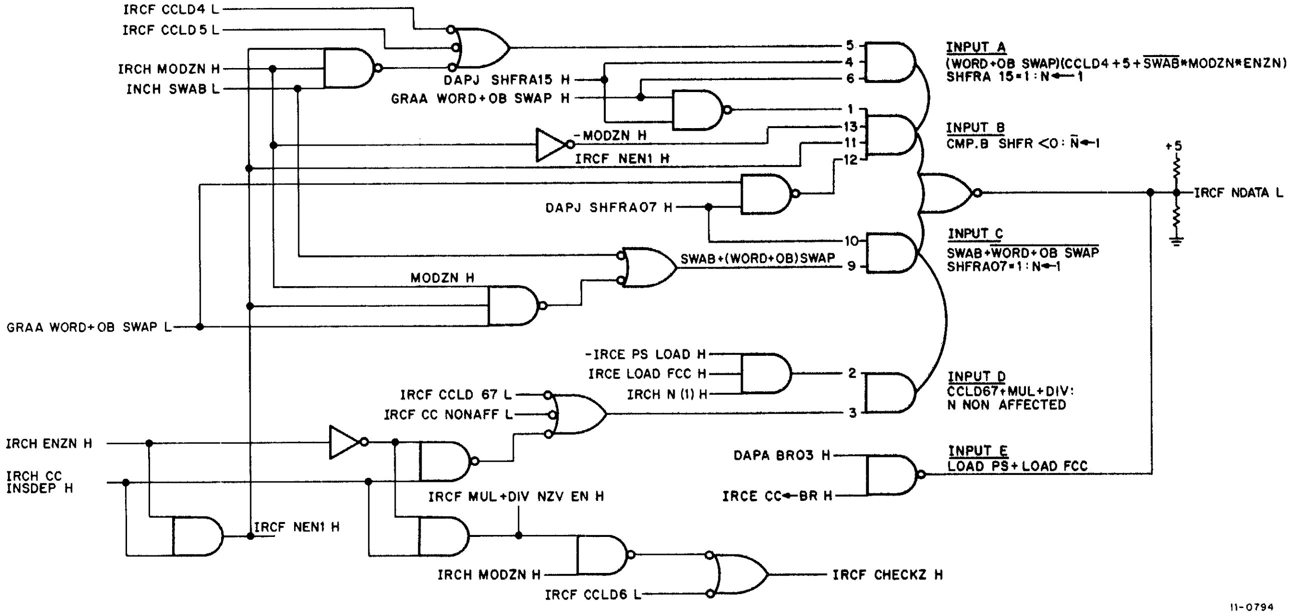


Figure 1-14 Sources of N Bit Data, Simplified Diagram

Table 1-12
C Bit Data Sources

Instruction	CC Control ROM			IRCF CDATA L Source
	CMOD1	CMOD0	ENC	
ROR.B, ASR.B	0	0	0	C ← AMX00 (VMOD0=1)

ROL.B, ASL.B	0	0	0	C ← AMX08 (WORD)
				C ← AMX08 (OB)

ASHC	0	0	1	C ← DR00
COM.B, NEG.B, SBC.B SUB	0	1	0	C ← -ALUCN
MUL	0	1	1	C ← -X
CLR.B, ADC.B TST.B CMP.B, ADD	1	0	0	C ← ALUCN
ASH	1	0	0	C ← AMX00
MFP, MTP, SXT INC.B, DEC.B MOV.B, BIT.B, BIC.B BIS.B, XOR	1	0	1	non-affected

DIV	1	1	0	C ← 1
				C ← 0 if -DR15

SWAB				C ←0
Condition-Code Load Signals				
IRCF CCLD4				C ← 0
IRCF CCLD5				C ← AMX15
IRCF CCLD6				non-affected
IRCF CCLD7				C ← ALUCN

The input B logic tests for CMP.B instructions. Under these conditions, if SHFRA15 is 0, the N bit is set, and if SHFRA15 is 1, the N bit is cleared. Input D covers all cases where the N bit is not affected by the current operation, and is therefore reloaded with the previous content, IRCH N(1) H. Input E allows IRCF NDATA L to be asserted by BR03 for load PS and load FCC functions. Table 1-13 summarizes the sources of N bit data.

1.5.7 Z Bit Data

The Z (Zero) bit of the PSW is set when the result of a processor operation is 0. The Z bit data that controls the condition code is generated by logic on drawings IRCF and GRAB.

Figure 1-15 is a simplified diagram of the logic that asserts IRCF ZDATA1 L and GRAB ZDATA2 L. These outputs are clocked into the Z1 and Z2 flip-flops, whose contents are ORed to provide the Z bit of the PSW condition code.

ZDATA1 Sources – The input gates that assert IRCF ZDATA1 L cover the special conditions that control the Z bit, independent of the SHFR outputs being equal to 0. For example, during the DIV instruction execution, MODZN and ENZN are both low and the Z bit is set. For the special case of the CMP.B instruction, the logic tests for the SHRF output = 1 condition to determine the Z bit. The other input gates that assert IRCF ZDATA1 L test for load PS or load FCC operations and operations that have no effect on the Z bit. Under the former conditions, the Z bit is loaded from BR02 and under the latter conditions, the Z bit is unchanged [Z(1)H controls ZDATA1]. These special conditions are summarized in Table 1-14.

Table 1-13
N Bit Data Sources

Instruction	CC Control ROM		IRCF NDATA L Source
	MODZN	ENZN	

CMP.B	0	1	N ← 1 if -SHFRA15 = 1
			N ← 0 if SHFRA15 = 1

DIV	0	0	non-affected
MUL	1	0	non-affected
all other instruction-dependent codes	1	1	N ← 1 if SHFRA15 = 1 (word or odd byte) N ← 1 if SHFRA07 = 1 (byte)
SWAB			N ← 1 if SHFRA08 = 1
Condition-Code Load Signal			
IRCF CCLD4			N ← if SHFR = 0
IRCF CCLD5			N ← if SHFR = 0
IRCF CCLD6			non-affected
IRCF CCLD7			non-affected

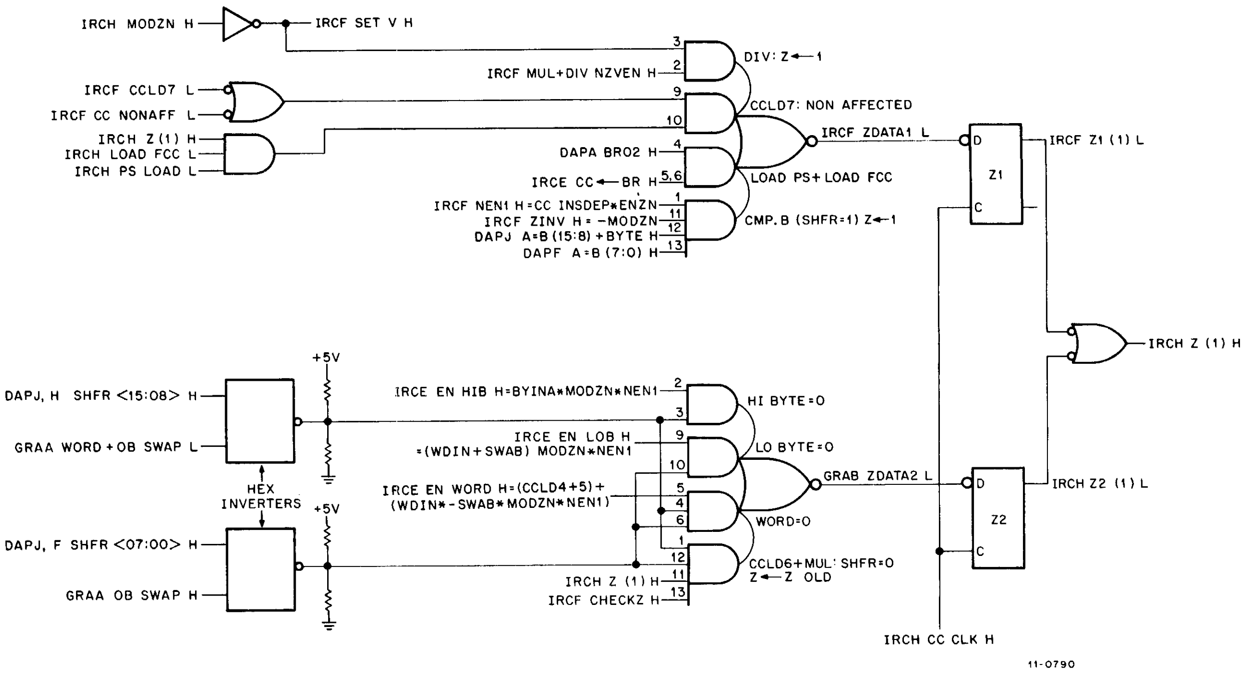


Figure 1-15 Sources of Z Bit Data, Simplified Diagram

Table 1-14
Z Bit Data Sources

Instruction	CC Control ROM		Z Data Source
	MODZN	ENZN	
CMP.B	0	1	Z ← 1 if SHFR = 1
MUL	1	0	Z ← 2(1)H if SHFR = 0
DIV	0	0	Z ← 1
SWAB			Z ← 1 if SHFR ⟨07:00⟩ = 0
all other instruction-dependent codes	1	1	Z ← 1 if SHFR = 0
Condition-Code Load Signals			
IRCF CCLD4			Z ← 1 if SHFR = 0
IRCF CCLD5			Z ← 1 if SHFR = 0
IRCF CCLD6			Z ← Z(1)H if SHFR = 0
IRCF CCLD7			non-affected

ZDATA2 Sources – The logic that generates GRAB ZDATA2 L tests the SHFR output for 0. The open-collector inverters function as 0 detectors for SHRF(15:08) and SHFR(07:00). The enabling inputs, IRCE EN HIB H, IRCE EN LOB H, and IRCE EN WORD H are used to test each byte of the SHFR separately, or together. The additional GRAB ZDATA2 gate tests the SHFR output word for 0 under CCLD6 or MUL conditions. If the SHFR output is 0, the previous Z bit condition, Z(1)H, controls the new Z bit.

1.5.8 V Bit Data

The V (overflow) bit of the PSW is set when a processor operation results in an arithmetic overflow. The logic that generates the V bit data is shown on drawing IRCE. The V bit is affected by two broad categories of instructions: arithmetic operations, and word or byte operations. The results of these operations and other special cases determine IRCE VDATA L. To simplify the description, arithmetic operations and special cases are grouped as VEN1 inputs. Word and byte operations are grouped as VEN2 inputs. Table 1-15 summarizes the V bit data sources of both groups.

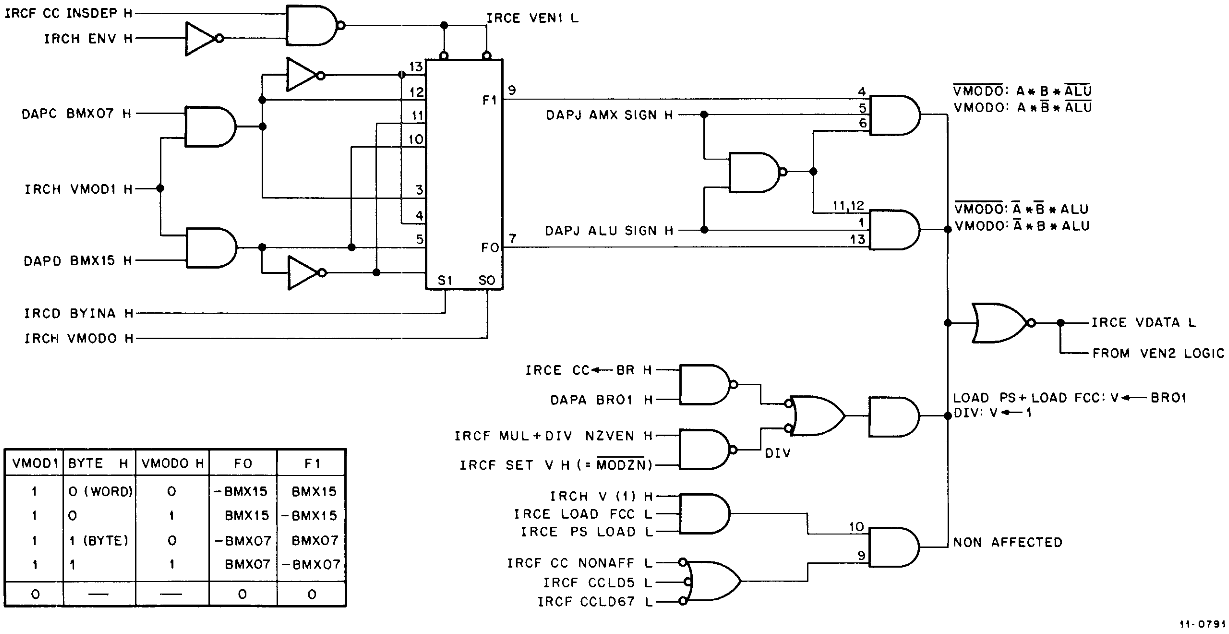
VEN1

Figure 1-16 is a simplified diagram of the V bit data sources that are grouped in the VEN1 category. A 74S153 Dual 4-Line-to-1-Line Multiplexer is used to select the most-significant BMX bit for the arithmetic operations that involve the B input.

Table 1-15
V Bit Data Sources

Instruction	CC Control ROM			IRCE VDATA L Source*
	VMOD1	VMOD0	ENV	
		VEN1		
INC.B, ADC.B	0	0	0	V ← -A*ALU15
DEC.B, SBC.B	0	1	0	V ← A*-ALU15
NEG.B, ADD	1	0	0	V ← A*B*-ALU15 + -A*-B*ALU15
SUB, CMP.B	1	1	0	V ← A*-B*-ALU15 + -A*B*ALU15
		VEN2		
MFP, MTP, SXT, CLR.B, COM.B, TST.B, MOV.B, BIT.B, BIC.B, BIS.B, MUL, ASH, ASHC, XOR	0	0	1	V ← 0
DIV	0	0	1	V ← 1
ROL.B, ASL.B	1	0	1	V ← SHFRA15 ∀ AMX15
ROR.B, ASR.B	1	1	1	V ← SHFRA15 ∀ AMX00
		Condition-Code Load Signals		
IRCF CCLD4				V ← 0
IRCF CCLD5		(VEN2)		V ← V old + (SHFRA15 ∀ AMX15)
IRCF CCLD6		(VEN1)		non-affected
IRCF CCLD7		(VEN1)		non-affected

*A = DAPJ AMX SIGN H
B = DAPD BMX15 H (word) or DAPC BMX07 H (byte)
ALU15 = DAPJ ALU SIGN H



VMOD1	BYTE H	VMODO H	FO	F1
1	0 (WORD)	0	-BMX15	BMX15
1	0	1	BMX15	-BMX15
1	1 (BYTE)	0	-BMX07	BMX07
1	1	1	BMX07	-BMX07
0	—	—	0	0

Figure 1-16 VEN1 Sources of V Data Bit, Simplified Diagram

These are NEG.B, ADD, SUB, and CMP.B, as indicated in Table 1-15. For these instruction-dependent codes, the CC CNTL ROM asserts IRCH VMOD1 H, which gates the BMX outputs to the multiplexer inputs, and IRCE VEN1 L, which enables the multiplexer. IRCD BYINA H selects BMX15 or BMX07 as the most-significant bit. IRCH VMODO H selects the BMX bit or its complement at each output, as shown on the multiplexer truth table in Figure 1-16.

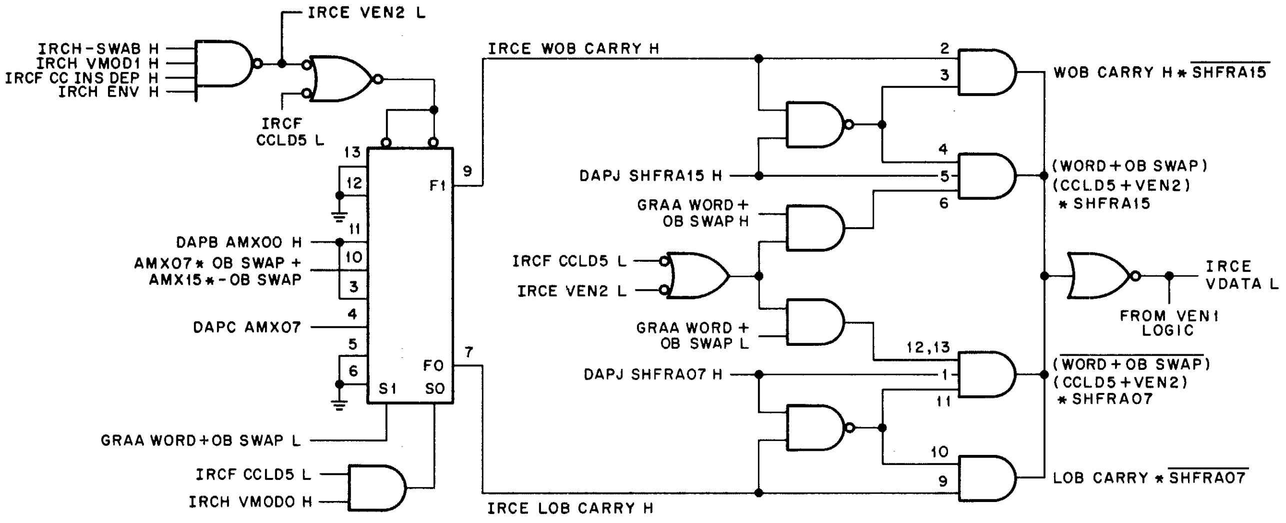
The notation on Figure 1-16 indicates the conditions and functions for which each AND gate input asserts IRCE VDATA L.

For INC.B, ADC.B, DEC.B, and SBC.B instruction-dependent codes, CC CNTL ROM output IRCH VMOD1 H is low. As a result, the BMX multiplexer outputs are always 0. For these instructions, B is eliminated from the source function, as listed in the source column of Table 1-15.

VEN2

Figure 1-17 is a simplified diagram of the V bit data sources that are grouped in the VEN2 category. A 74S153 Dual 4-Line-to-1-Line Multiplexer selects the most-significant AMX bit for the word, odd-byte, or byte operations. The multiplexer truth table is shown on Figure 1-17. The multiplexer is only enabled by CCLD5, or those instruction-dependent codes for which the CC CNTL ROM asserts IRCH VMOD1 H and IRCH ENV H. As indicated in Table 1-15, these instructions include ROL.B, ASL.B, ROR.B, and ASR.B. For these instructions, the notation on Figure 1-17 indicates the conditions and functions for which each AND gate input asserts IRCE VDATA L.

For the majority of the instructions included in the VEN2 group of Table 1-15, VMOD1 is low. As a result, the AMX multiplexer is not enabled and none of the AND gate inputs will be enabled because IRCE VEN L is not asserted. Therefore, processing these instructions clears the V bit.



WORD OR ODD BYTE SWAP	VMODO *$\overline{\text{CCLD5}}$	IRCE LOB CARRY	IRCE WOB CARRY
YES	NO	0	AMX07 (ODD BYTE) AMX15 (WORD)
YES	YES	0	AMX00
NO	NO	AMX07	0
NO	YES	AMX00	0

11-0792

Figure 1-17 VEN2 Sources of V Data Bit, Simplified Diagram

CHAPTER 2
DATA PATHS

This chapter describes the Data Paths of the KB11-C Processor. The Data Paths consist of the logical elements that execute the data manipulations required by the Control section. The inputs to and the outputs from the Data Paths, as well as the Data Paths themselves, are described in this chapter.

All the elements of the Data Paths logic are controlled by the microprogram ROM; a separate field of the ROM output word controls each of these elements. These fields, the values that they can assume, and the function executed by the logic unit, are listed on the block diagram, Figure 2-1.

The Arithmetic and Logic Unit (ALU), performs most of the arithmetic and all of the logic (AND, OR, EXCLUSIVE-OR) functions required by the instruction set (Paragraph 2.1.1).

The ALU is the input to the Program Counter (PC) and to the Shifter (SHFR). The PC (Paragraph 2.1.3) consists of two registers (PCA and PCB) and is used both to keep track of the next program instruction and as an auxiliary register during data manipulation. The SHFR is the input to the General Registers (GR) and to the Bus Register. The SHFR transfers data from the ALU or from the PCB. The ALU data may be either unchanged, shifted one bit to the right, or byte-swapped (Paragraph 2.1.2).

The General Registers consist of two identical copies of 16 registers (00–17$_8$): one copy consists of the General Source (GS) registers, the other consists of the General Destination (GD) registers. Both of these copies are written at the same time and are identical (Paragraph 2.1.4).

The Source Register and Destination Register multiplexers (SRMX and DRMX, Paragraph 2.1.5) transmit data from the GRs (including the PC or GR7) to the Source Register (SR) and to the Destination Register (DR).

The SR and DR (Paragraphs 2.1.6 and 2.1.7), as their name implies, are used for source and destination address and operand storage. In addition to this function, they are used as storage during certain instructions, such as MPY, DIV, ASH and ASHC. The SR cannot change data, but the DR can shift either right or left.

The Shift Counter (SC) is used only for instructions that require multiple shifting: MPY, DIV, ASH and ASHC. A value is loaded into the SC, which counts to zero; at this time the instruction is completed. The DR is the input to the SC (Paragraph 2.1.8).

The logic elements described above, plus the BR, and the Constant Multiplexers (K0MX and K1MX) are the inputs to the ALU, via two multiplexers (AMX and BMX). These two multiplexers correspond to the A and B inputs of the ALU. AMX, BMX, K0MX and K1MX are described in Paragraph 2.1.9.

The Bus Register Multiplexer (BRMX, Paragraph 2.2.1) receives data from all inputs to the Data Paths and selects one for storage in the Bus Registers (BR and BRA, Paragraph 2.2.3) and, during an instruction fetch, into the Instruction Registers (IR and AFIR, Paragraph 2.2.4).

The inputs to the BRMX are the Cache, the SHFR, the Unibus via the Bus Buffer Register, and the Internal Data Bus (INTD, Paragraph 2.2.2).

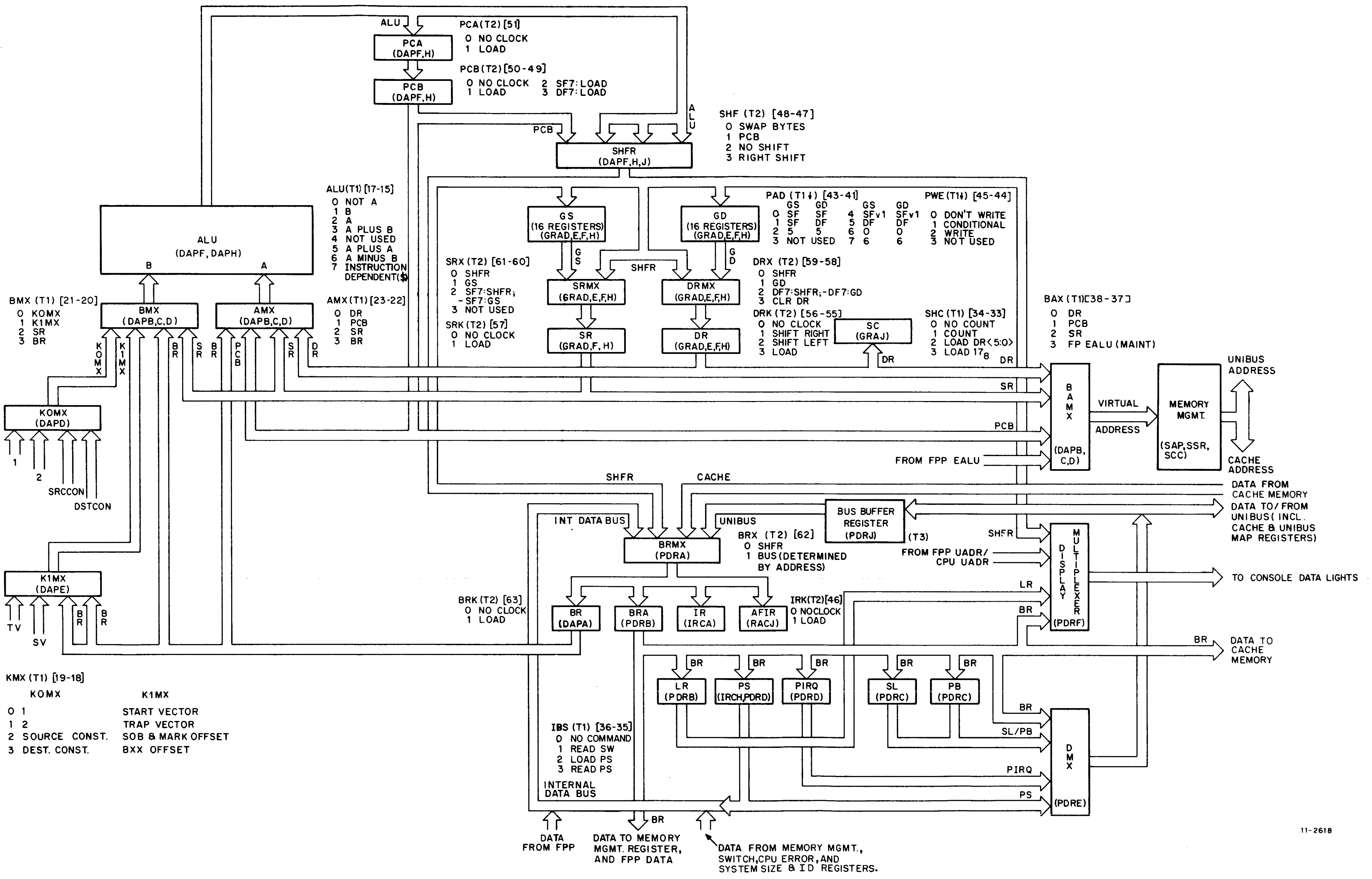


Figure 2-1 Block Diagram
Data Paths

The outputs of the processor Data Paths select and supply address, data and display information:

1. The Bus Address Multiplexer (BAMX, Paragraph 2.3.1) selects the virtual address for transmission to Memory Management from either the DR, the SR, or the PC.

2. The (Unibus) Data Multiplexer (DMX, Paragraph 2.3.2) selects the source of data to the Unibus from the BR or from the Control Registers (Chapter 3).

3. The BRA supplies data directly to the Cache, the Memory Management registers, the Floating Point Processor and the Control Registers (Paragraph 2.3.3).

4. The Display Multiplexer is controlled by the Data Display selection switch on the Console and selects the source of the Console data display from the SHFR, the FPP and CPU ROM Address Registers, the Light Register or the BR (Paragraph 2.3.4).

2.1 DATA MANIPULATION

Data manipulation is done mainly by the logic elements, shown in the top-half of the Data Paths Block Diagram, Figure 2-1.

The ALU is the most complex of these elements and is the only one that can combine two operands. It is the first one described. Its outputs are input to the PC or to the SHFR, from where they may be routed to the General Registers, to the SRs and DRs and back to the ALU via the A and B multiplexers.

2.1.1 Arithmetic and Logic Unit (ALU)

The primary data processing element in the KB11-C (the only element that can combine two operands to form a result) is the Arithmetic and Logic Unit (ALU). The ALU can perform a variety of arithmetic operations on two variables (such as addition or subtraction) and can perform a variety of logical operations on one or two variables, such as complementing or ANDing. The specific operation performed at any time is selected by the processor control on the basis of the microprogram word and the current instruction. The manipulated operands are selected by two multiplexers, one for each of the ALU inputs. The operands can be the contents of the SR, the DR, the BR, the PCB, or one of several numbers generated by the constant multiplexers.

The output of the ALU is gated either into PCA or into the SHFR, from which it can then be routed to any of the General Registers, or to the SR, the DR, or the BR (and the IR, although this path is not used). All of these destinations for manipulated data are internal to the processor; when data is transferred out of the processor, it must go through the BRA. When the ALU outputs are routed to the PC, the signal paths do not pass through the SHFR; this means that when shift or byte-swap operations are attempted with register 7 as the destination, the data that enters the PCA is unchanged. For example, an ASR PC instruction does not shift the PC but does set the condition code as would an ASR.

2.1.1.1 Description of ALU – Refer to drawings DAPF and DAPH. The ALU does most of the data manipulation in the processor. It operates on two 16-bit words of data and a carry input to produce one 16-bit word of data and a carry output. When the M input is high, the ALU operates in the logical mode; when this signal is low, the ALU operates in arithmetic mode. The carry signals are not active when the ALU is operating in the logical mode. Drawing DAPF shows the low byte and DAPH shows the high byte of the ALU.

The 16-bit ALU is implemented with four 74S181 4-bit Arithmetic Logic Units. Each 74S181 includes look-ahead carry generation for the four bits. A second level of look-ahead carry generation is provided by the 74182-1 Carry Generator. The carry-propagate (P) and carry-generate (G) outputs of each 74S181 (except the most-significant four bits) are connected to the corresponding inputs of the 74182-1, and the carry outputs of the 74182-1 are connected to the appropriate carry inputs of the ALUs. The least-significant bit carry input is controlled by GRAA ALUC H, based on the output of the subsidiary instruction-dependent ALU control ROM.

The ALU can perform any one of 16 logical functions (each output bit is dependent only on the corresponding input bits) or any one of 16 arithmetic functions (each output is dependent on the corresponding input bits and on a carry propagated from less-significant bits). The selection of a particular function is controlled by five signals from the GRA module which select the mode (arithmetic or logical) and the function. The **KB11-C** uses only ten of the possible 74S181 functions. These ten functions are listed at the bottom of drawing DAPF.

The low order byte of the ALU is controlled by the S0 – S3 inputs (DAPF LS0 H – DAPF LS3 H) and the M input (DAPF LM H). The high order byte is similarly controlled by DAPH HS0 H – DAPH HS3 H and DAPH HM H. All of these signals are derived from GRAA ALUS0 L – GRAA ALUS3 L and GRAA ALUM L.

In addition to the data and carry outputs, each ALU element has a comparator output, which indicates (if the ALU is in subtract mode) that the two inputs are equal. These outputs, which are open-collectors, are wire-ANDed for each data byte to generate equality signals that are used in forming the condition codes.

DAPF A = B(7:0) H indicates that the inputs to the low data byte are equal.

DAPF A = B(15:0) L indicates that the inputs to the entire word are equal. DAPH BUS A = B(15:8) H is the wired-AND of the A = B outputs for the high-byte ALUs on drawing DAPH.

Four signals that are used in the generation of the Condition Codes are derived from the ALU:

1. DAPJ AMX SIGN H is the sign of the A input to the ALU. This signal corresponds to AMX15 if the processor is operating on word data, or to AMX07 if the processor is operating on byte data.

2. DAPJ ALU SIGN H is the sign of the ALU output; it is taken from ALU15 for word data or from ALU07 for byte data.

3. DAPJ A = B(15:8) + BYTE H indicates either that the high data byte is all 0s or that the processor is operating on byte data. This signal is used in determining whether all the active data is 0s for the Z condition code.

4. DAPJ ALUCN L is the carry output of the active portion of the ALU; it takes the carry output from the high byte for word data or the carry output from the low byte for byte data. This signal is used to generate the Carry (C) condition code.

2.1.1.2 ALU Control – During each machine cycle, the ALU performs the function that is specified by the ROM ALU control bits [RACC UALU(2:0) H]. The signals that actually control the ALU (and also the SHFR) operations are shown on schematic GRAA.

If the UALU bits equal 0 – 6, the control signals are independent of instructions being executed. If these bits equal 7, the control signals depend on the instruction code. In this last case (instruction dependent), the notation "$ALU" appears on the Flow Diagrams.

The ALU control signals generated on the GRA module are:

GRAA ALUS(3:0) L	(ALU S0 – S3 control)
GRAA ALUM L	(ALU mode control)
GRAA ALUC H	(Carry in)

GRAA ALU INSDEP L controls the two 74S158 multiplexers that select the source of these ALU control signals. GRAA ALU INSDEP L is low when the UALU bits equal 7 (A inputs), and high when the UALU bits equal 0 – 6 (B inputs).

Non-Instruction Dependent Control

The ALU control field in the main microprogram ROM is a 3-bit field that controls the values of six control signals. There is not a one-to-one relationship between the ROM bits and the control signals, and not all possible combinations of control signals can be generated. Each control signal is the result of decoding the ROM bits.

RACC UALU0 and UALU2 are inverted by the multiplexer and generate GRAA ALUS3 and ALUS2, respectively. If UALU = 1 or 6, the output of the 74S64 at the lower-left of GRAA goes high and GRAA ALUS1 goes low; for other values of UALU, ALUS1 is high. If UALU = 3, the B0 input to the multiplexer is high and ALUS0 is low.

The M bit is asserted when UALU = 0 or 1; GRAA MODE H goes high and ALUM L goes low. The carry bit is generated when UALU = 6 by GRAA CIN L, which goes low and causes GRAA ALUC H to go high.

These control signals are all inverted on DAPF and DAPH and input to the ALU. Table 2-1 shows the operation performed by the ALU for each value of the UALU field, and the state of the control signals at the 74S181.

Instruction-Dependent Control
When the ALU control signals are instruction-dependent, each of the six signals is controlled by a separate output signal from the subsidiary ALU control ROM, shown on drawing GRAA. The ROM inputs [IRCH SUBROMA(4:0) H] are described in Chapter 1, Paragraph 1.5.

When UALU = 7, the multiplexer S0 inputs are low and the A inputs are selected. Two of the ALU select signals, GRAA ALUS0 and ALUS1, take on the value of the ROM outputs. The other two, GRAA ALUS2 and ALUS3, are forced high when the SWAB instruction is being executed. The SWAB instruction does not have a unique ROM word, and uses the same word as the ASL instruction with some of the control signals modified in this manner. Refer to the ALU Control ROM Map, shown on drawing GRAK.

The ALUM (mode control) signal is taken directly from the ROM, except when the SXT instruction is executed with a negative operand [IRCH N(1) H is high] or when both GRAA ROMM and ROMC are high (GRAA CDEP L).

In the case of SXT and a positive operand [IRCH N(1) H low], GRAA ROMM is high, ROMC is low; this forces GRAA ALUM low, DAPF LM and DAPH HM high, which puts the ALU in the logic mode. DAPF LS0 - LS3 (and DAPH HS0 - HS3) are respectively L, L, H, H and the ALU output is 0 (refer to the ALU table on DAPF). In the case of a negative operand [IRCH N(1) H high], GRAA ALUM is high, which puts the ALU in the arithmetic mode. All other control signals being unchanged, the ALU output is a 2's complement minus 1 (all 1s).

GRAA ROMM and ROMC are both high for the ROL, ROLB, ADC, ADCB, SBC and SBCB instructions. In this case, GRAA CDEP L is low and the ALU is put in the arithmetic mode instead of in the logic mode.

Table 2-1
Non-Instruction-Dependent ALU Control Signals

UALU	Operation	Control Signals					
		DAPF or DAPH					Negation of GRAA ALUC H
		LS3H HS3H	LS2H HS2H	LS1H HS1H	LS0H HS0H	LMH HMH	
0	not A	L	L	L	L	H	
1	B	H	L	H	L	H	
2	A (plus carry)	L	L	L	L	L	
3	A plus B (plus carry)	H	L	L	H	L	L
4	not used						
5	A plus A (plus carry)	H	H	L	L	L	L
6	A-B	L	H	H	L	L	H
7	instruction-dependent	Instruction Dependent					

The ALU C (Carry-in) signal is modified for two classes of instructions. The DIV and ASHC instructions operate on 2-word operands, and the instruction-dependent state is one that shifts the two words left. The carry-in must take on the state of the most-significant bit of the less-significant word. For the ADC on ROL instructions, a carry insert signal is generated if the C bit is set; for the SBC instruction, the signal is generated if the C bit is cleared. This data-dependent carry generation is controlled by the assertion of both ROMM and ROMC.

GRAA SGNEX MOVB is generated when a MOVB instruction is being executed. This instruction is used to extend the sign of the byte into the high byte when the destination is a General Register.

GRAA WORD + OB SWAP L and H indicate that the significant SHFR outputs include the high byte, and the sign of the output is bit 15 (rather than bit 7).

2.1.2 Shifter (SHFR)

The output of the ALU is input to the program counter (PCA) and to the SHFR. The inputs to the SHFR include, in addition to the ALU, the output of PCB.

The SHFR can perform right-shift or byte-swap operations on the data, or substitute the contents of the PC for the ALU outputs. In many cases, where an instruction is performed for an odd-byte destination operand, the data manipulation required by the instruction is completed in the ALU and the transfer of the result to the odd-byte data lines is performed in the SHFR, all during one machine cycle.

In addition to its data manipulation (shifting and byte swapping) activity, the SHFR is used as a routing element. When General Register 7 (the PC) is transferred to the SR or to the DR, PCB is routed through the SHFR, to the SRMX or DRMX, then to the SR or DR.

The output of the SHFR goes to the General Registers, GS and GD, to the SRMX and DRMX, to the BRMX and to the display multiplexer - where it provides the Data Paths display data.

2.1.2.1 Description of SHFR - The SHFR is a four-input multiplexer that provides unshifted, right-shifted and byte-swapped outputs from the ALU inputs. It accepts PCB as the fourth input. Left-shift operations are performed in the ALU by using the A plus A mode. The sum of A added to A is equivalent to the product 2A, which in turn is equivalent to shifting A (as a binary number) one bit to the left.

Bits (00:06) and (08:14) of the SHFR are similar, and are shown on drawing DAPF and DAPH.

Special operations are required in the SHFR for the most-significant bit of each byte. The SHFR logic for data bits 7 and 15 are shown separately on drawing DAPJ.

BITS 00:06 AND 08:14

Refer to Figure 2-2, which shows a typical SHFR bit 00:06 or 08:14.

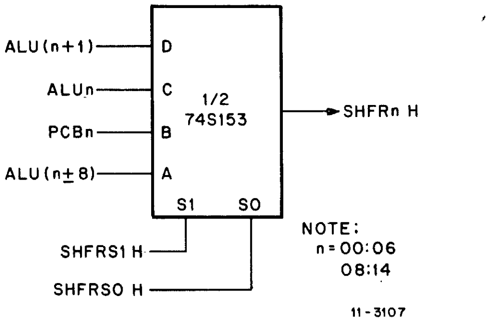


Figure 2-2 Typical SHFR Bit

When a byte swap is required, the A inputs are selected, and ALU(08:14) are switched to the outputs of SHFR(00:06), and ALU(00:06) to the outputs of SHFR(08:14). Inputs B switch the PCB to the multiplexer outputs. Inputs C transfer ALU(00:06) and (08:14) to SHFR(00:06) and (08:14) (no shift). A right shift is executed by using input D, which transfers ALU (n+1) to SHFR n (for example, ALU05 to SHFR04).

BITS 07 and 15
Refer to drawing DAPJ. The most significant bit of the shifter is SHFR 15. The shifter inputs are similar to the inputs for other shifter bits when the byte-swap (A) or unshifted ALU inputs (C) are selected. However, the input used for the right-shift mode is dependent on the instruction being executed.

For some shift operations, such as ASR and ASRB, the sign of the data word is replicated. This is done by routing ALU15 (the most-significant, or sign, bit) to the right-shift inputs of both DAPJ SHFR 15 and DAPH SHFR 14. For right rotate (ROR and RORB) instructions and multiply instructions, this procedure is modified by forcing a second level 2-input 74S157 multiplexer to select GRAJ SHFR DATA H instead of DAPH PCB 15 H. The signal GRAJ SHFR DATA consists in this case of the carry (C) bit and the P/class instruction decode for the rotate instruction. For the multiply instruction, the input is used to extend the sign of the result during the calculation and to correct the sign on the cycle, if necessary. In this last case, it is high if the instruction is an I/class, and either the SR is greater than 0 during an instruction-dependent cycle, or the contents of the SR are negative (SR15 1) during a non-instruction dependent cycle.

The shifter logic for data bit 7 must operate the same as the normal bits for word data, and as the most-significant bit for byte data. The right-shift input must be able to receive one of three values; ALU08 for word data; ALU07 for byte shifts (if not a rotate instruction); or the Carry (C) bit for an RORB instruction. This is accomplished by multiplexing the C bit with the PCB input and forcing the SHFR to accept input B for an RORB instruction; for any other byte shift, the SHFR is forced to accept input C, the no shift input, so that SHFR07 and SHFR07 both receive ALU07. SHFRA15 and SHFR15 signals and SHFRA07 and SHFR07 signals are logically identical and appear only for additional loading capacity.

GRAB Z DATA2 L detects all 0s at the SHFR output. Depending on the operation being performed, either the entire word of data or only one byte of data may be significant. For word data, both wired-AND circuits must detect all 0s. For normal byte operations, only the low byte (SHFR07 – SHFR00) must be all 0s. During operations on odd bytes, or during a SWAB instruction, only the high byte is tested. A fourth input, enabled by IRCF CHECKZ H, is used when the final result is two words, to clear the 0 (Z) bit if the second word does not contains all 0s. If the second word is all 0s, the Z bit retains the previous value. Thus, only if both words are all 0s will the Z bit be set.

2.1.2.2 Shifter Control – The SHFR is controlled by DAPF SHFRS0 and SHFRS1 H, which are inverted from GRAA SHFRS0 and SHFRS1 L. These signals, in turn, are generated by the same subrom that controls the ALU, and they are instruction-dependent when the ALU control signals are. Refer to Paragraph 2.1.1.2.

GRAA SHFRS0 and SHFRS1, when instruction-dependent, take on the value of the subrom output, except in the case of the ASRB, ROROB, NEG and NEGB instructions if the destination mode is not 0, and in the case of the SWAB instruction. In both of these cases, DAPF SHFRS0 and SHFRS1 are forced low by GRAA SWAP L.

2.1.3 Program Counter (PCA and PCB)

The Program Counter (PC) provides the address of the next instruction to be fetched. The PC is implemented as two 16-bit registers, PCA and PCB.

PCA accepts data only from the ALU; this data is clocked in at T5 by DAPJ CLKPCA H when the PCA ROM bit =1. The output of PCA goes only to PCB, and is the only input to PCB.

PCA is clocked into PCB at T1 by DAPJ CLKPCB H when the PCB ROM bits =1, 2 or 3: 1 is an unconditional load; 2 loads if the source field =7; 3 loads if the destination field =7, unless the instruction is I/class and the UPWE00 ROM bit is high.

(I/class instructions are those that cause a high output of the ITCH R (I CLASS) output of the instruction decode subrom. They are listed in the R (I/CLASS) column of the table on IRCJ).

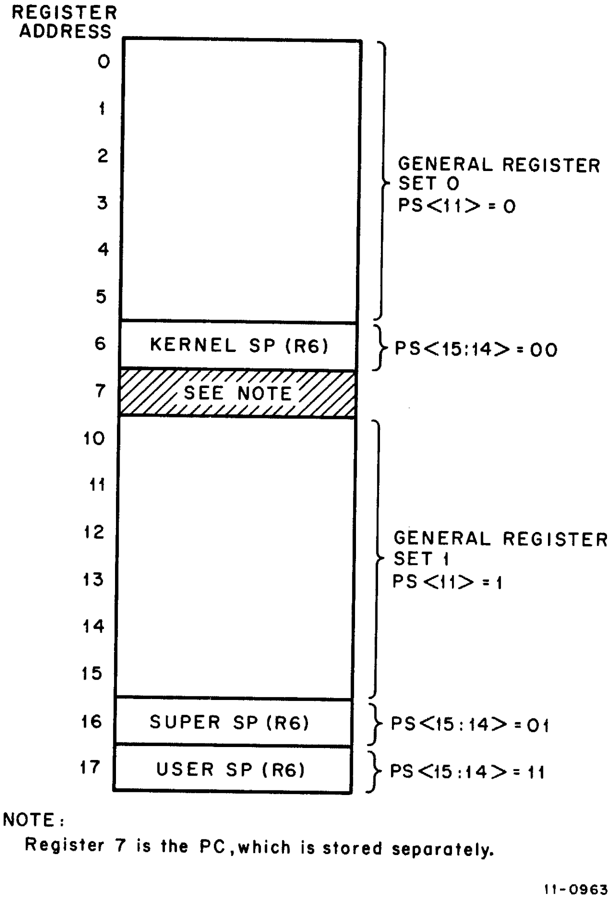
2.1.4 General Registers

In all instructions that transfer data, each address reference specifies one of eight General Registers. The specific register (of the 16 in the KB11-C Processor) used for each reference depends both on the value of the 3-bit register specification and on the processor state, as represented by the contents of the Processor Status (PS) word.

Two of the eight General Registers that can be specified in the instruction code are also used by the **KB11-C** as special-purpose registers. If the register specification has a value of 7, it specifies the Program Counter (PC). This always refers to the hardware PC register described in Paragraph 2.1.3. If the specification has the value 6, it specifies the hardware Stack Pointer (SP) register.

One of three hardware registers, within the General Register data storage elements, is selected in this case, depending on the processor mode: register 6 if the processor is in Kernel mode, 16 if it is in Super mode, or 17 if in User mode. If the register specification has the value 0 – 5, one of two registers is selected, depending on the register set selection bit (bit 11 in the PS word).

Figure 2-3 illustrates the General Register selection in the **KB11-C** Processor. Figure 2-4 shows the format of the Processor Status word (PS).



NOTE:
Register 7 is the PC, which is stored separately.

Figure 2-3 General Register Storage in GS and GD Storage Elements

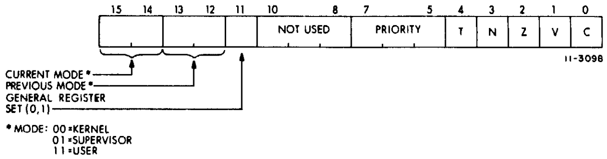


Figure 2-4 Processor Status Word

Each of the 16 General Registers is duplicated. The duplication allows the processor to access more than one register at a time. Each General Register, with the exception of register 7, is implemented by two copies in the two General Register storage elements.

The General Source (GS) registers include 16 registers allocated as shown in Figure 2-3. The General Destination (GD) registers contain 16 registers used in an identical manner. When data must be written into a General Register, it is written into both cop ies to ensure that all attempts to read the data will read the same value. However, by specifying different register addresses to the GS and GD storage elements, it is possible to read the contents of a different register from each. This feature is used primarily in reading the contents of the two registers specified by double-operand instructions.

Whenever the General Registers, as a group, serve as a data source, the PC (register 7) can be selected as one of the General Registers. This is accomplished by selecting the PCB input to the SHFR, and allowing the source or destination multiplexer to select the SHFR input, if register 7 is selected, and the GS or GD input if any other register is selected.

Refer to schematics **GRAD–GRAH**. The General Registers are implemented in two sets of four 3101A 64-bit random-access memories that are arranged in sixteen 4-bit words. Each General Register is made up of one word from each of four memories, and the same word selection signals are sent to all four memories for one copy of the registers. A different set of selection signals can be sent to the second copy of the registers while reading, but not when data is being written.

Data is written when the W input is low. The write enable signals are GRAC GRWE LOB L for the low order byte, and GRAC GRWE HIB L for the high-order byte. The conditions for these signals are explained in a table on GRAC.

Individual registers are selected for reading and writing by GRAC GDA (0:2) H and by GRAC GSA (0:2) H, all four of which go to the A0 - A2 inputs to the 3101As. The register sets are selected by GRAC GDREG SET1 H and GSREG SET1 H, which go to the A3 inputs to the 3101As.

General Register Selection

Source and Destination Address Multiplexer [GRAC GSA(0:2), GDA(0:2)] - The microprogram selects the sources of the scratch pad addresses. The microprogram includes a 3-bit PAD field that selects one of seven sets of sources; the value of 3 in the PAD field is not used. Some of the sources are constants, and are generated by +3 V and 0 V inputs to the GDAM and GSAM multiplexers; others are taken from the IR source and destination register specifications of the instruction. Table 2-2 shows the multiplexer inputs used for each PAD value. Table 2-3 shows the values of these inputs.

The multiplexers are disabled when PAD =6; GSA (0:2) and GDA (0:2) are low in this case.

Table 2-2
Multiplexer Input Selection
GSAM and GDAM

PAD	GSAM	GDAM
0	A	A
1	A	B
2	C	C
3	not used	
4	A	A
5	B	B
6	GS and GD MX disabled	
7	D	D

Table 2-3
Multiplexer Input Values

Input	Value	
	Bits 1 and 2	Bit 0
A	Source Field [IR(07:08)]	If IR06=1, high. If IR06=0, low, unless current mode is User and the source field = 6 or 7. If PAD=4, same as above, but GRAC PLUS 1 is ORed with IR06 to force an odd register address. Used only during MUL, DIV and ASHC.
B	Destination Field [IR(01:02)]	IF IR00=1, high. If IR00=0, low if the console is not active; or if the destination field is not =6; or if PS15=0 (Kernel or Super current mode) and the instruction is other than MFP or MTP with destination mode 0; or if PS13=0 (Kernel or Super previous mode) and the instruction is MFP or MTP with destination mode 0.
C	GSA(2:0) and GDA(2:0) = 5	
D	If PS15=0, GSA(2:0) and GDA(2:0)=6 (Register 6, Kernel or Super) If PS15=1, GSA(2:0) and GDA(2:0)=7 (Register 6, User)	

General Register Set Selection (GRAC GDREG SET 1 and GSREG SET 1) - The most-significant bit of the scratch pad address selects which General Register set is used. This selection is, in general, done by the multiplexer; in several cases, the processor forces the selection of General Register Set 1. Note that these multiplexers are always enabled.

Table 2-4 shows the multiplexer inputs selected for each PAD value.

Table 2-4
Multiplexer Input Selection
GSREG and GDREG SET 1

PAD	GSREG SET 1	GDREG SET 1
0	A	A
1	A	B
2	C	C
3	not used	
4	A	A
5	B	B
6	C	C
7	D	D

GRAB SRC SET 1 L and DST SET 1 L are, respectively, the A and B inputs to both source and destination multiplexers.

Both gates are asserted (low) when the Console is not active, PS11 is asserted, and registers 0 - 5 are specified by the source [IR(06:08)] or destination [IR(00:02)] fields of the current instruction; registers 0 - 5 are selected if not both IR08 and 07 (for the source field) or IR02 and 01 (for the destination field) are asserted.

Set 1 is also selected when the Console is not active, PS14 is asserted (Super or User modes), and register 6 is specified by the instruction source or destination fields. This, in conjunction with the GRAC multiplexer outputs, forms address 16. If PS15 is asserted, the A input to GRAC GDA0 and GSA0 is forced high, thus generating address 17 (GRAC PLUS 1). If the instruction is an MFP or an MTP, and UPEW00 =1 (conditional), and the destination field =6 or 7, and the mode is User or Super, and the Console is not active, GRAB DST SET1 L is also asserted.

The C input to the Set 1 multiplexers is PS11, which defines the register set.

The D input to these multiplexers is PS14(1)L which, when asserted (low), specifies User or Super modes.

The output of these multiplexers, when low, causes the selection of General Register Set 1 through the GRAC GDREG SET1 H and GSREG SET1 H OR gates.

The two other inputs to the OR gates cause the selection of SET1:

1. During a Console operation, bit 3 of the address selects the Register Set and is clocked into IR03; it is then input to the OR gates to select the proper set.
2. In the case of an MFP or MTP instruction with destination mode 0 and destination field =6, if UPWE00=1 (conditional) and PS12=1 (previous User or Super modes), set 1 is also selected. In an MFP instruction, the source is always specified in the field normally designated as destination. The destination is the current mode stack.

2.1.5 Source and Destination Multiplexers (SRMX and DRMX)

The SRMX and DRMX select the input to the Source and Destination Registers (SR and DR). Refer to drawing GRAD.

The select inputs to these multiplexers are GRAC SRMX SEL L and DRMX SEL L, which are controlled by the SRX and DRX ROM bits and by IRCB SRCF 7 L.

When the SRX and DRX bits =0, the SHFR is selected as the input to the SR and DR. When SRX and DRX =1, the General Source and Destination registers (GS and GD) are the SR and DR inputs. If SRX and DRX =2, the inputs are either the SHFR, if the Source or Destination fields =7, or the GS and GD if this is not the case. SRX =3 is not used; DRX =3 clears the DR at GRAJ TP(3:5), which is a flip-flop set by T3 and reset by T5.

2.1.6 Source Register (SR)

The Source Register (SR) performs two major functions. It is the output buffer for the General Registers when addressed as the SR in an instruction, and it provides temporary storage during the source data-fetch operations.

All output from the GS registers must be transferred through the SR. When the PC is selected as a source register, the data from the PCB is routed through the SHFR and the SRMX to the SR. From the SR, data can be routed anywhere in the processor through the ALU inputs, or the contents of the SR can be used as an address for external data transfers through the BAMX. The SR is also used as a temporary storage register during transfers of data within the processor; e.g., when the old PC and PS are being stacked during an interrupt or trap service sequence, the SR holds the vector address.

The SR is used as a data storage element for intermediate results during instruction execution. The register and operand group instructions, such as multiply, divide, and the arithmetic shifts, use the SR to hold both operands and results.

The outputs of the SRMX are connected directly to the inputs of the SR and are clocked by T1 if enabled by the microprogram bit RACA USRK H. The outputs of the SR are routed to the ALU input multiplexers and to the bus address multiplexer. Bit 0 of the SR is also sent to the IRC module for use in one of the microprogram address generation circuits, the C Fork, for odd-byte source branches.

The output of the SR is checked for two conditions: SR ≤ 0 and SR = +1, by GRAE SR LEQ ZERO H and SR EQ ONE L. The two flip-flops are clocked by the same signal that clocks the SR. They are both set if GS(01:15) = 0.

GRAE SR LEQ ZERO H is asserted if both flip-flops are set and GRAD SR00 H is low (SR=0) or if GRAH SR15 L is asserted (SR is negative).

GRAE SR EQ ONE L is asserted if both flip-flops are set and GRAD SR00 H is asserted (SR=+1).

2.1.7 Destination Register (DR)

In addition to performing two functions similar to the major functions of the SR, the Destination Register (DR) also operates as a data manipulation element; specifically, the DR is used as a left or right shift register during register and operand instructions such as ASH, ASHC, MUL, and DIV.

All output from the GD registers (and from the PC, when it is selected as a destination register) must be through the DR. Data from the DR can be routed anywhere in the processor through the ALU, or used as an address in external data transfers through the BAMX. To transfer the contents of either the SR or the DR to an external data storage location, the data must first be transferred from the SR or DR through the ALU to the BR, and then from the BR to the Cache, the Unibus, or the Internal Data Bus.

The DR is used as a control register and to accumulate the less-significant part of the result during register and operand instructions such as multiply, divide, or the arithmetic shifts. The DR is also the source for data to be loaded into the Shift Counter (SC) register.

Refer to GRAD through GRAH. The DR can be loaded with a left shift of one bit, a right shift of one bit, or no shift. The shift inputs are used when the processor must operate on two words of data at the same time (for example, during a multiply or divide instruction) and the operation includes shifting. The type of loading is determined by RACA UDRK(00:01), as shown on GRAD. During a right shift, DAPF ALU00 is loaded into GRAH DR15. During a left shift, DAPJ LEFT DATA is loaded into GRAD DR00; DAPJ LEFT DATA is high when both DAPJ COUT15 H (the ALU carry out) and the instruction is I/class. This input is used during the DIV instruction. When no shift is required, DRMX(00:15) are loaded into DR(00:15).

The DR is cleared when the DRMX control bits UDRX(00:01)=3.

At T1, when UDRK(00:01)=3 (load DR), DRMX00 is clocked into the GRAB OBD (Odd-Byte Destination) flip-flop. When set, this flip-flop indicates that the destination field contains an odd byte address.

2.1.8 Shift Counter (SC)

The Shift Counter [GRAJ SC(00:05)] is used to count the repetitive cycles of data manipulation in the multiply (MUL), divide (DIV), arithmetic shift (ASH), and arithmetic shift combined (ASHC) instructions. The SC can be loaded either with the six less-significant bits of the DR (for ASH or ASHC instructions) or with a constant, 17(8), (for MUL or DIV instructions). The SC is controlled by the RACC USHC(00:01) ROM bits. The outputs of the SC are used in the Branch Conditions logic on RACK.

The SC consists of two 74191 counters and associated logic. They are loaded with the value present at the D inputs when the LOAD input is low. The 74191 counts on the positive transition of the clock signal, if the ENABLE input is low. The counter counts down if the DN input is high, and counts up if DN is low. The MAX/MIN output goes high when the outputs are all high (=1111), and the count direction is up (DN=low), or when the outputs are all low (=0000) and the count direction is down (DN=high). The R/CLK (ripple clock) output goes low when MAX/MIN is high and CLK is low. The R/CLK from the low order SC clocks the high order SC. If RACC USHC(01:00)=0, the SC is inoperative.

If USHC=1, the ENB input is low and one clock pulse is generated at GRAJ TP(3:5) H.

If USHC=2, the complement of DR(05:00) is loaded with the sign extended to the two unused high order bits of the SC.

If USHC=3, the eight bits of the counter are loaded with 1s. This is used to count to 16(10) (=17_8) during MUL and DIV. In this case, only the four low order bits [SC(03:00)] are counted.

Refer to Figure 2-5. When 17_8 is loaded, SC05L is low, and the counter is made to count up, since SC05L is input to both DN inputs. At the first clock pulse, SC(00:03) goes to all 0s (1111+0001). Neither MIN/MAX nor R/CLK are generated at this time, and SC(04:05) stay high. Each clock pulse increments the contents of SC(00:03) by 1. When their value equals 1111, MIN/MAX goes high, and since SC(04:05) are still high, GRAJ SC=0 L is asserted. This occurs on the sixteenth clock pulse.

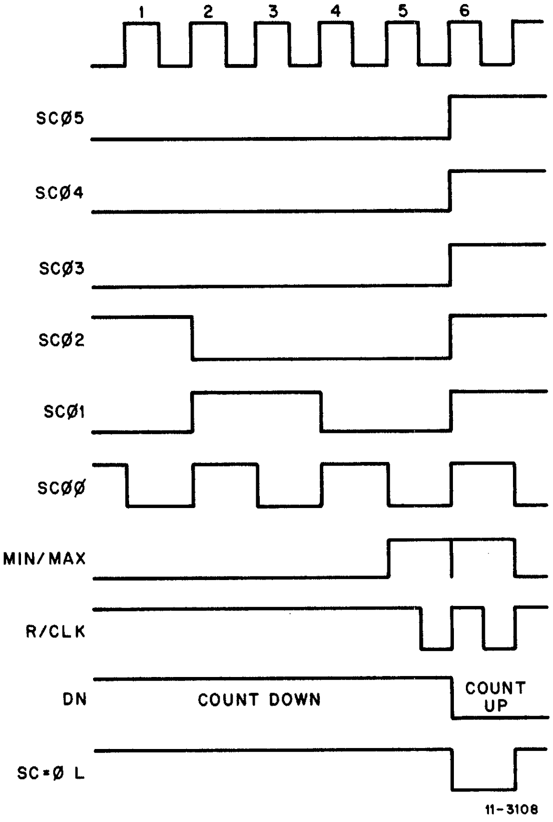


Figure 2-5 SC Loaded With 00101

Refer to Figure 2-6. When an ASH or ASHC specifies a right shift, bits (0:5) of the instruction word contain a negative value. This causes a positive value to be loaded into the SC (SC05=0), and the counter will count down (GRAJ SC05 L = DN are high). Assume that a 6-bit shift is desired: -6 in 2's complement, or 11010, is entered into bits (5:0) of the instruction word and then loaded into the DR. The 1's complement of this value, or 00101, is the number loaded into SC(05:00). Since the DN input is high, successive clock pulses cause the counter to count down to 00000. At this time, MIN/MAX goes high, but since SC05 is low, GRAJ SC 0 L is not asserted. At the next clock pulse, the sixth, R/CLK is asserted. Since the counter is still counting down, all five SC bits change from 00000 to 11111. GRAJ SC05 L and the DN input both go low, which defines count up. MIN/MAX stays high, SC04 and SC05 are high, causing GRAJ SC 0 L to be asserted, thus ending the count.

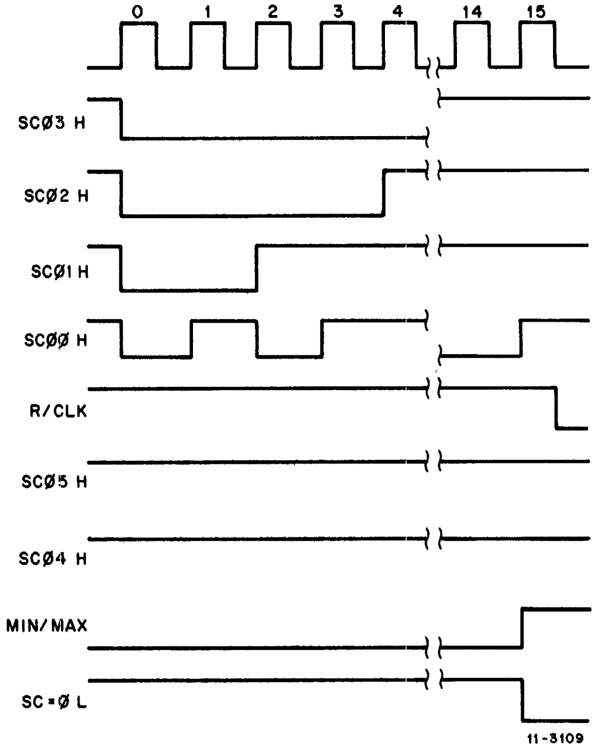


Figure 2-6 SC Loaded With 17_8

2.1.9 ALU Inputs

The A multiplexer (AMX) is the "A" input to the ALU. It can select one of four signals: DR, SR, PCB, or the Bus Register (BR).

The B multiplexer (BMX) is the "B" input to the ALU. It can select the SR, the BR or one of two constant multiplexers, K0MX or K1MX.

General information on these inputs is listed in Table 2-5.

2.1.9.1 A Multiplexer (AMX) - The A multiplexer (DAPB AMX00 H - DAPD AMX15 H) is controlled by RACC UAMX(01:00) and selects one of four registers for input to the A operand of the ALU. The values of RACC and the registers selected are listed in the table on drawing DAPB.

2.1.9.2 B Multiplexer (BMX) - The B multiplexer (DAPB BMX00 - DAPD BMX15 H) selects the B input to the ALU. It is controlled by RACC UBMX(01:00) H. Table 2-6 shows the outputs of the BMX for the several values of UBMX.

Table 2-5
ALU Input Multiplexers

Multiplexer	Output To	Input From	Type of Input
AMX	A input of ALU	source register destination register bus register program counter	variable operand variable operand variable operand variable operand
BMX	B input of ALU	source register bus register KOMX K1MX	variable operand variable operand constants constants and sign-extended operands
KOMX	BMX	1 2 source constant destination constant	fixed constant fixed constant generated constant generated constant
KIMX	BMX	trap vector start vector BR (SOB & MARK) BR (branch)	generated constant fixed constant shifted and sign-extended operand shifted and sign-extended operand

Table 2-6
BMX Output Selection

BMX	RACC UBMX(01:00) H			
	00	01	10	11
00	KOMX00	0	SR00	BR00
01	01	K1MX01	01	01
02	02	02	02	02
03	03	03	03	03
04	0	04	04	04
05	0	05	05	05
06	0	06	06	06
07	0	K1MX07	07	07
*08	KOEX	K1EX*UKMX00	08	08
*09	KOEX	K1EX	09	09
*10	KOEX	K1EX	10	10
*11	KOEX	K1EX*UKMX00	11	11
*12	KOEX	K1EX	12	12
*13	KOEX	K1EX	13	13
*14	KOEX	K1EX	14	14
*15	KOEX	K1EX	SR15	BR15

*Note: If GRAA SGNEX MOVB L is asserted, KOEX H becomes the output of BMX(15:08) H.

Sign Extension – When RACC UALU(2:0) H = 7 (ALU instruction dependent), and the instruction is MOVB (IRCB MOVB H is high), GRAA SGNEX MOVB L is low. This forces the two signals that control BMX(15:08) high (DAPD BMXS1 HIB L and BMX S0 HIB L), thus putting DAPD K0EX H on the high order byte BMX output line. K0EX takes on the value of BR07 when the BR is selected (UBMX=3), or that of SR07 when the SR is selected (UBMX=2).

2.1.9.3 Constant Multiplexer 0 (K0MX) – Constant Multiplexer 0 [DAPD K0MX(03:00)] supplies values required for incrementation of ALU operands. The K0MX is controlled by RACC UKMX(01:00) H.

When UKMX=0, a constant of 1 is generated. When UKMX=1, a constant of 2 is generated, except in the case where FRMJ ADDR INC L (request by the FP11 for an address increment) is not asserted and TMCE FC H is asserted. FC (Floating Point Condition) is asserted when the Bus Condition bits (BSC)=4, signifying that the processor is executing a memory operation for the FP11.

When UKMX=2, a constant of 1 is generated if IRCC SRCCON-1 H is asserted. A constant of 2 is generated if IRCC SRCCON-1 H is asserted. The conditions for these functions are shown on drawing IRCC and are mutually exclusive. They normally indicate an auto-increment or auto-decrement addressing mode for the source register.

When UKMX=3, constants of 1, 2, 4, or 10 may be generated by IRCD DSTCON-1 (or 2, 4, or 10) H. Increments of 4 or 10 are only used for FP11 instructions. The conditions for these functions are shown on drawing IRCD.

DAPD K0EX H is described in Paragraph 2.1.9.2, B Multiplexer (Sign Extension).

2.1.9.4 Constant Multiplexer 1 (K1MX) – Constant Multiplexer 1 [DAPE K1MX(07:01) H and K1EX H] generates vector addresses and program counter offsets. The K1MX is controlled by RACC UKMX(01:00) H.

Table 2-7 shows the output of the BMX for the several values of UKMX when the K1MX is selected (UBMX=1).

When UKMX=0, DAPE SV(07:02) H are selected. This is the start vector, which is selected in ROM state 100 (PUP.00 on Flows 12) during the power up sequence. The address may be selected either in the range of 000 000 to 000 174(8), or in that of 173 200 to 173 374(8), depending on the jumper for SV07. This is due to the logic for DAPE BMX08 and 11 combined with the K1MX circuitry, which extends the sign to all high order bits except bits 08 and 11.

The trap vector (TV) is used to select a new PC and PS following a trap operation. The trap vectors for a variety of internal conditions are defined by the logic in the lower-left corner of the drawing. The chart on DAPE defines the specific vector for each condition. If none of these conditions is present, but the processor is doing a trap operation, the trap vector is set to 4. This occurs for non-existent memory references, memory parity errors, odd address errors, fatal stack violation errors, and executing the Halt instruction in User or Supervisor modes of operation. The K1MX constants for EMT and TRAP instructions are one-half their assigned values. This is because they are executed by the same machine states (Flows 12) that cause the vector for reserved instructions to be left shifted (so that vector 4 forms vector 10).

Table 2-7
BMX Output From K1MX

Bit	BMX RACC	RACC UKMX(01:00) H			
	UBMX=1	00	01	10	11
00	0	0	0	0	
01	K1MX01	0	TV01	BR00	
02	↑ 02	SV02	↑ 02	↑ 01	
03	03	↑ 03	↓ 03	02	
04	04	04	TV04	03	
05	05	05	TV05*07	↓ 04	
06	↓ 06	↓ 06	TV06	BR05	
07	K1MX07	SV07	TV05*07	0	BR06
08	K1EX*UKMX00	0	0	0	BR07
09	K1EX	SV07	0	0	BR07
10	K1EX	SV07	0	0	BR07
11	K1EX*UKMX00	0	0	0	BR07
12	K1EX	SV07	0	0	BR07
13	K1EX	SV07	0	0	BR07
14	K1EX	SV07	0	0	BR07
15	K1EX	SV07	0	0	BR07

The third input to K1MX, BR(07:00)H, is used for the offset in SUBTRACT 1 AND BRANCH (SOB), and MARK instructions. This offset is always in full words and is always a positive quantity that is subtracted from the PC in the ALU. Because all PDP-11 Systems use byte addresses, the offset, as it appears in the instruction, must be multiplied by 2 to generate the proper value to be subtracted from the PC. This is done by shifting the 6-bit offset 1 bit to the left. For example, BR00 is the input to the multiplexer for bit 01. The BR is used because it contains the same value as the Instruction Register (IR) at the time of the PC modification, and is directly-accessible to the data path logic.

The fourth input to K1MX is used for the offset in successful branch instructions. The branch offset can be either positive or negative; the value taken from the instruction is first multiplied by 2 (shifted left) and then sign-extended, and the resulting 16-bit number is added to the PC. The branch offset can have values from $+127_{10}$ to -128_{10} words; BR (07:00) provide the offset and the left shift provides word (rather than byte) addresses.

2.2 INPUTS TO PROCESSOR DATA PATHS

The Processor Data Paths receive data through the Bus Register Multiplexer (BRMX) from the Cache Memory, the Console (Switch Register), the Memory Management registers, the optional Floating Point Processor and the Unibus. The Unibus input is buffered by PDRJ D(15:00) H, the Bus Buffer Register, which is clocked at every TIGD T3 L.

The BRMX also has an input for internal data from the SHFR. The most generally used path from the SHFR to the ALU is through the BRMX and the BR.

The BRMX is the input to the two Bus Registers (BR and BRA) and to the two Instruction Registers (IR and AFIR).

2.2.1 Bus Register Multiplexer (BRMX)

All data input to the processor is routed through PDRA BRMX(15:00) H; in addition to the external data from the Unibus, the BRMX also accepts inputs from the Cache Memory, the SHFR, and the Internal Data Bus.

The four inputs to the BRMX are:

1. PDRJ D(15:00) H (Bus Buffer Register clocked each T3 from the Unibus lines);
2. PDRA INT D(15:00) H (Internal Data Bus)
3. DTML CDM(15:00) H (Cache Memory data)
4. DAPF-DAPJ SHFR(15:00) H (Shifter output).

Refer to Figure 2-7. These signals are selected by PDRA BRMX S(1:0) H. The SHFR is selected when RACA UBRX H is low, making S1 and S0 both high.

The other three inputs can only be selected when UBRX is high.

The Cache is selected when TMCF SEL MEM L is low; the address is a Cache address, an interrupt pause is not in progress, and the Internal Data Bus is not selected.

The Internal Data Bus is selected if TMCF SEL INT L is low. One of three conditions may cause this: an internal register is being addressed, or the IBS00 ROM bit is asserted (read Switch Register or read PS), or the BCT(02:00) ROM bits = 1 (read Floating Point data).

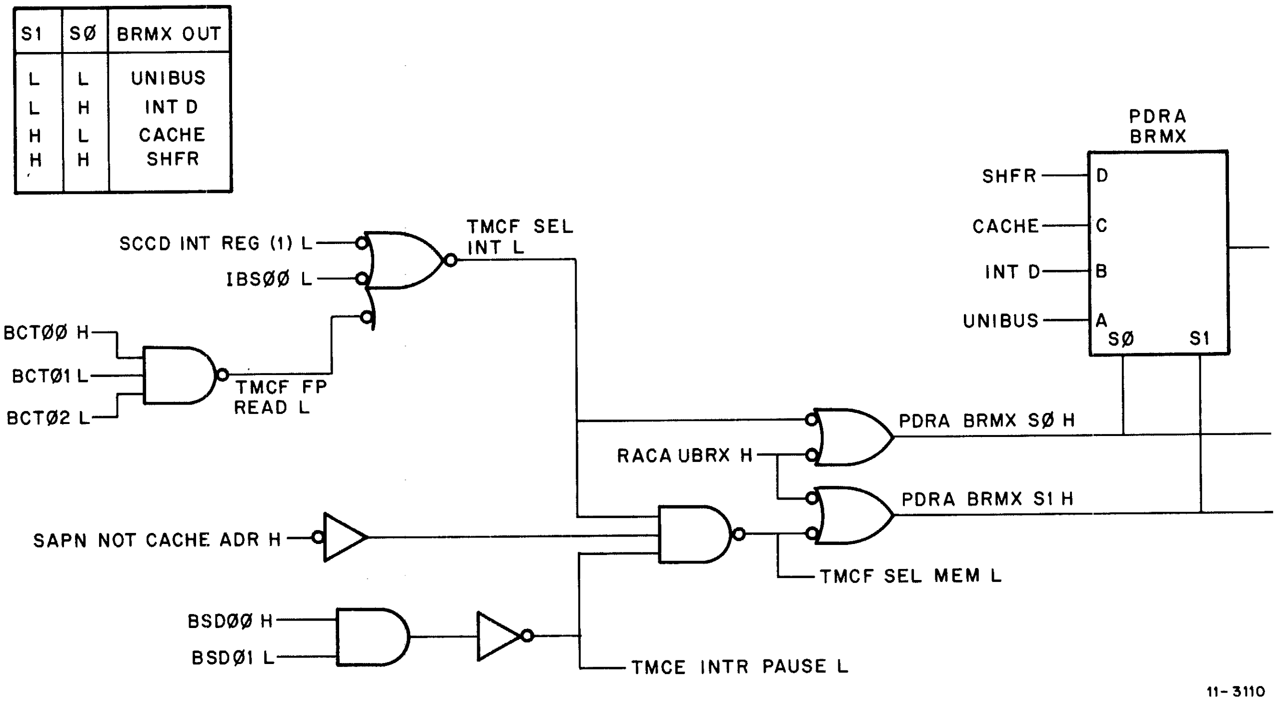
The Unibus is selected when UBRX, TMCF SEL MEM L, and TMCF SEL INT L are all high.

The BRMX is the input to both Bus Registers (BR and BRA) and to both Instruction Registers (IR and AFIR).

2.2.2 Internal Data Bus (INTD)

The Internal Data Bus [PDRA INT D(15:00) H] is a wired-OR bus that transmits the following data to the BRMX:

1. Switch Register (from Console)
2. Memory Management Registers (MMR3 to MMR0 and APR, which is a multiplexer that can select either a PAR or a PDR)
3. System ID and System Size Registers



S1	SØ	BRMX OUT
L	L	UNIBUS
L	H	INT D
H	L	CACHE
H	H	SHFR

Figure 2-7 BRMX Selection, Simplified Schematic

4. Processor Error Register (TMCD TRAPS TO 4)
5. Processor Status Word (PS)
6. Floating Point Processor Data [FXPD DOMX(15:00)].

Figure 2-8 is a block diagram of the Internal Data Bus. The data put on the bus is a function of the IBS (Internal Bus) and BCT (Bus Control) ROM bits. Refer to schematic TMCF. TMCF GET OFF H is asserted when the IBS field equals 1 (Read Switches), 2 (Load PS) or 3 (Read PS), or when the BCT field equals 1 (Read Floating Point Processor Data). TMCF GET OFF H is inverted on SSRJ and becomes SSRJ GET OFF L.

When BCT=1, data from the FP11 is enabled onto the bus and all the Memory Management inputs are disabled by TMCF GET OFF.

When IBS=1, 2 or 3, the Memory Management inputs are also disabled. IBS=1 selects the Switch Register. IBS=3 selects the PS.

When IBS=0 and BCT is not equal to 1, the Memory Management inputs are enabled. The selection of the register that is to be put on the INT D bus is made by register address decoding in Memory Management. Four schematic drawings (SSRJ, SCCH, SCCM and SCCN) show the Memory Management inputs to the Internal Bus. These inputs are:

1. MMR0 - MMR3
2. APR (PAR/PDR multiplexer)
3. System Size and ID registers
4. TMCD traps to 4 error register
5. Switch Register

One of these inputs is put on the Internal Bus if SSRJ GET OFF L is high, and if the operation is a read (SSRJ C1 B L not asserted).

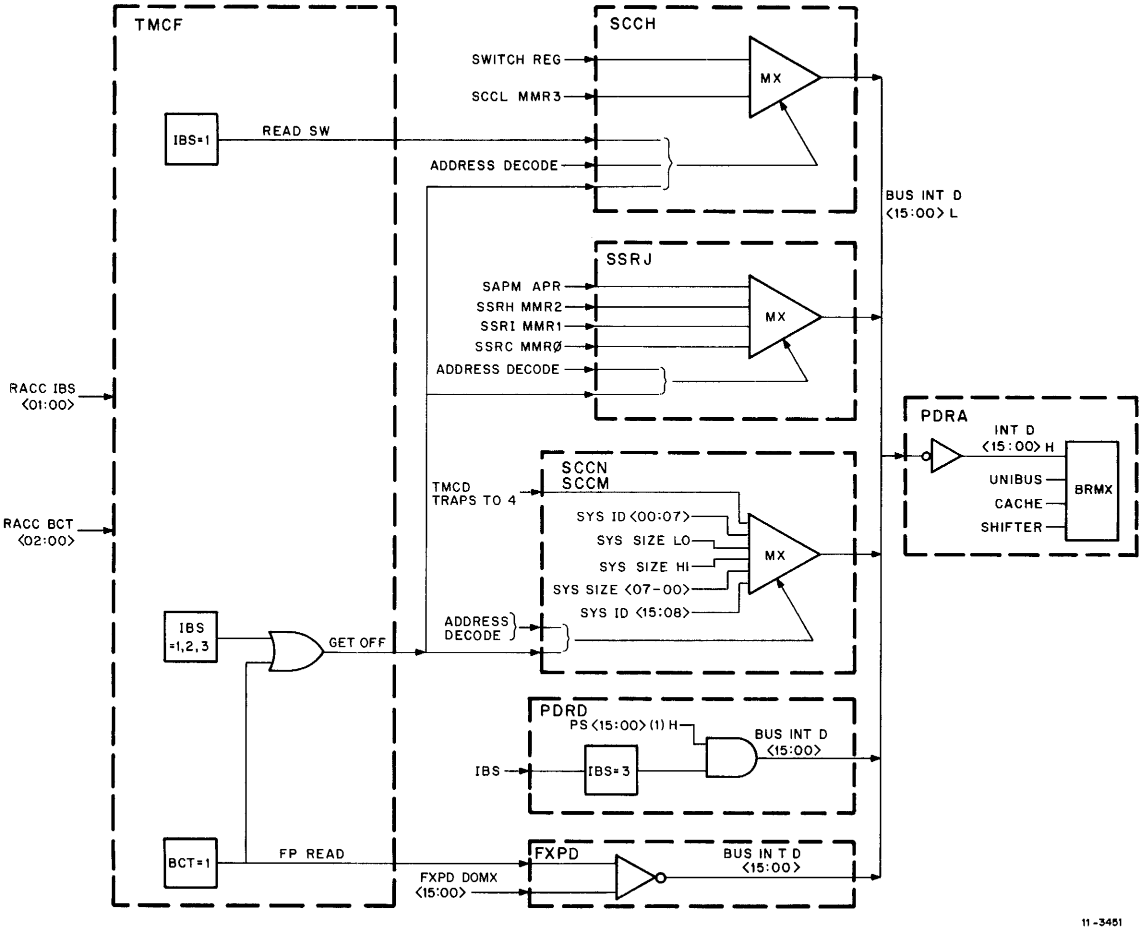


Figure 2-8 Internal Data Bus Block Diagram

Address decode determines which one of the inputs goes onto the bus.

2.2.2.1 SSRJ Multiplexer - The inputs to the multiplexer on SSRJ are MMR0, MMR1, MMR2 and the APR multiplexer (PAR or PDR). This multiplexer is enabled when SCCC INT REG B H is asserted and SCCC MMR3 is cleared in addition to GET OFF and C1.

Input select signals are SCCC MMR REG (1) H (MMR0, 1, 2), SSRH VA(02:01) L (virtual address bits 02:01) and SCCD APR REG L. VA(02:01) define which MMR is being addressed.

2.2.2.2 SCCH Bus Output - The Switch Register [SCCJ SWR(15:00) H] is transmitted from the Console to Connector J2 on SCCJ. It is multiplexed with MMR3 to make up the second Memory Management input to the Internal Data Bus.

Since MMR3 consists of only five bits (00, 01, 02, 04 and 05), only these bits need be multiplexed.

The MMR3 input is selected when SCCC READ MMR3 L is asserted.

The Switch Register is selected by SCCC SW REG (0) H when the reference is an explicit one and by TMCF READ SW L if the reference is implicit. This last signal is asserted when the ROM IBS field is equal to 1.

2.2.2.3 SCCM Multiplexer - The Multiplexer on SCCM transmits the following data on BUS INTD(07:00) L:

1. The System ID Register, bits (07:00),
2. The CPU Error Register (refer to Chapter 3), which consists of TMCD ILL HALT H, ODD ADRS H, CACHE NXM H, UBUS TIMEOUT H, YEL TRAP H and SL RED ERR H, and
3. The two System Size Register low-order bytes.

The Multiplexer is enabled by SCCD INTD REG L. Address decode signals select the output signal and, in conjunction with SCCC C1 B L and SSRJ GET OFF L, enable the output drivers.

2.2.2.4 SCCN Multiplexer - The high-order bytes of the System ID Register [SCCN SYS ID(15:08) H] and the System Size Register are gated onto the Internal Bus on SCCN by their respective address decode signals and by GET OFF and the negation of C1.

2.2.3 Bus Registers (BR and BRA)

The Bus register consists of two slightly different registers, the BR and the BRA.

The BRMX is the input to both BR and BRA. This last register, however, also accepts the parity bits from Cache Memory (DTML HI BYTE PAR H and LO BYTE PAR H). These bits appear on the BRA outputs as PDRB HI PAR H and LO PAR H and are used only to generate PDRH IND HI PAR H and IND LO PAR H, which transmit byte parity information to the Console indicators.

The BR outputs are designated DAPA BR(15:00) H and DAPA BR14 L. The high outputs are the inputs to the AMX, the BMX and the K1MX. DAPA BR14 L is an input to RACK BRCAB 05 L.

The BRA outputs are called PDRB BR(15:00) A H. They are also inverted as PDRB BR(15:00) B L. They are the inputs to the Control Registers (LR, PS, PIRQ, SL, PB), the DMX, the Display Multiplexer, Cache Memory, Memory Management and the FPP.

The BR and the BRA are clocked by TIGA CLK BR H and CLK BRA H, during the 15 ns of the duration of TIGC TPB L, when RACA UBRK H (load BR) is high and TIGA GATE BR (1) L is low. This last flip-flop is set at the rising edge of TPB L when the output of the OR gate is high. This always occurs at T1 (refer to Chapter 4).

2.2.4 Instruction Registers (IR and AFIR)

When an instruction is fetched from an external data storage location, the data word enters the processor through the Bus Register Multiplexer (BRMX), and is loaded into the BR. To retain the instruction word for decoding during the execution of the instruction, while releasing the BR for other data transfers that may be required during the execution of the instruction, the outputs of the BRMX are simultaneously loaded into the instruction register [IRCA IR(15:00)] and into the A Fork Instruction Register [RACJ AFIR(15:00)].

The IR and AFIR are clocked only during data transfers that fetch instructions. The BR is clocked during every external data transfer that brings data into the processor. Both IR and AFIR are clocked by TIGC T1 or T1B if RACA UIRK H is asserted (Load IR).

The IR is used for decoding circuits which operate the subsidiary ROMs, the program ROM B and C Forks, and a variety of instruction class selectors. The instruction decoding logic is shown on the control section block diagram, Chapter 1. The AFIR is used only by the program ROM A Fork.

2.3 PROCESSOR DATA PATHS OUTPUTS

The output of the Data Paths is routed through one of four logic units:

a. The Bus Address Multiplexer (BAMX) selects the source of the Unibus address

b. The Display Multiplexer selects the source of the console data display

c. The Data Multiplexer selects the source of Unibus data

d. The Bus Register (BRA) supplies data directly to the Cache Memory, the Memory Management registers and the optional Floating Point Processor.

2.3.1 Bus Address Multiplexer (BAMX)

The Bus Address Multiplexer (DAPB BAMX00 H to DAPD BAMX15 H) accepts as inputs the DR, PCB and SR registers, as well as an input, used for maintenance purposes only, from the FP11 Floating Point Processor. Its output is the program virtual address, which is the input to Memory Management, which in turn generates the physical address for the Cache and the Unibus.

The BAMX output is selected by RACB UBAX(01:00), as shown on the table on drawing DAPB.

2.3.2 Unibus Data Multiplexer (DMX)

Refer to drawing PDRE. The Processor data output to the Unibus is BUS D(15:00) L, which consists of DEC 8881 bus drivers. The input to these drivers are the Data Multiplexer (DMX), and UBCA CPBSY B H, which gates the DMX outputs onto the Unibus. CPBSY generates BUS BBSY L during a Unibus transaction (refer to Chapter 5).

The inputs to the DMX (data outputs to the Unibus) are:

a. The Bus Register (BRA), which is used as the data output of the processor to Unibus devices. BRA is always selected during a processor DATO.

b. The Control Registers: PS (Processor Status word), SL (Stack Limit), PIR and PIA (Program Interrupt), PB (Program Break). When explicitly addressed (by Unibus address), these registers are read by the program from the Unibus during a processor DATI.

c. During any DATI other than those during which the processor reads the Control Registers, the output of the DMX is 0. This is because the data is coming from a Unibus device and the processor data lines must not be asserted.

The high order byte of the DMX corresponds to BUS D(15:08) and is enabled by TMCD HI BYTE EN H; the low order byte corresponds to BUS D(07:00) and is enabled by TMCD LO BYTE EN H. When these signals are not asserted, the corresponding outputs of the DMX are not asserted (low). In the case of the Control Registers (PS, SL, PIR and PIA, PB), one or the other, or both, of these signals are asserted when an internal address is decoded (SCCE INTERNAL ADRS H) by Memory Management and a Unibus transaction has been started (UBCA MSYN SET H). Both signals are asserted in the case of the BR (DATO = TMCD C1 B L).

The select signals (TMCD DMX S1 H and S0 H) are enabled by UBCA MSYN SET H and the negation of TMCD C1 B L (=DATI). The combination of select signals for each register is determined by register address decoding on drawing SCCE. If none of the Control Registers are selected, both select signals are low and the BR is selected.

During a DATO, both DMX S1 H and S0 H are low (C1 L is low) and the BR is selected.

Table 2-8 shows the selection of data outputs to the Unibus.

2.3.3 Bus Register A (BRA)

PDRA BR(15:00) A H transmits data to the Cache Memory write multiplexer CDPE WRITE MUX(15:00) H, to which the other input is Unibus data from the Unibus map [MAPA DATA(15:00) H].

The BR is also the input to the Memory Management registers, and the data input to the Floating Point Processor.

2.3.4 Display Multiplexer

The Display Multiplexer [PDRF DISP(15:00) H] selects the input to the Console data display [KNLA DISP(15:00) H].

The multiplexer select signals (PDRF DISPS1 L and S0 L) are the inversion of PDRH DISP DATA SEL1 H and SEL0 H, which in turn are the encoded outputs of the Console Data Display switch (KNLD DISP DATA SEL1 H and SEL0 H).

Table 2-9 shows the register displayed for each switch position.

Table 2-8
Data Output to Unibus

Unibus Output	SCCE INT ADRS H	UBCE MSYN SET H	TMCD					PDRE DMX	
			C1*	HI BYTE EN H	LO BYTE EN H	DMX S1 H	DMX S0 H	Input	Byte
PS	H	H	DATI	H	H	H	H	A	HI, LO
SL	H	H	DATI	H	L	L	H	C	HI
PIR PIA	H	H	DATI	H	H	H	L	B	HI, LO
PB	H	H	DATI	L	H	L	H	C	LO
BR	L	H	DATO	H	H	L	L	D	HI, LO
NONE	L	H	DATI	L	L	L	L	None	None

*NOTE: TMCD C1 B L low = DATO, high = DATI.

Table 2-9
Display Register Selection

Switch Position	KNLD DISP DATA SEL		Register Displayed
	1H	0H	
BUS REGISTER	L	L	BR(15:00)
DATA PATHS	L	H	SHFR(15:00)
DISPLAY REGISTER	H	L	LR(15:00)
μADRS FPP/	H	H	FRMA/B CRAR(7:1)
CPU	H	H	RACD RAR(7:1)

CHAPTER 3
PROCESSOR CONTROL REGISTERS

The KB11-C Processor contains registers which control processor operations or provide information relative to these operations. These registers, which are listed below, are described in this chapter (in order of ascending addresses):

Address	Register
17 777 570	Switch and Light Registers
17 777 760	Lower Size Register
17 777 762	Upper Size Register
17 777 764	System ID Register
17 777 766	CPU Error Register
17 777 770	Microprogram Break Register
17 777 772	Program Interrrupt Request Register
17 777 774	Stack Limit Register
17 777 776	Processor Status Word

Information on Memory Management, Unibus Map and Cache Registers are contained in Sections IV through VI of this manual.

3.1 SWITCH REGISTER (SWR) AND LIGHT REGISTER (LR)

The Switch Register is the output of the Console switches. It shares address 17 777 570 with the Light Register, whose input is the BR and whose only output is the Console Display indicators through the Display Multiplexer when the Console Data display switch is in the DISPLAY REGISTER position.

The SWR is read-only and the LR [PDRB LR(15:00)] is write-only. They are both described in Section III of this manual.

3.2 LOWER SIZE REGISTER

This read-only register [SCCN SYS SIZE(21:14), bits 13:06 are all 1s] specifies the memory size of the system. It indicates the last addressable block of 32 words in memory (the high order byte indicates the number of 8K blocks of available memory minus 1). It is used by Memory Management to determine the validity of an address. It is read on the Internal Data Bus (INTD) at address 17 777 760 (bit 0 is equivalent to bit 6 of the Physical Address). Refer to Section IV, Memory Management.

3.3 UPPER SIZE REGISTER

This register is an extension of the system size, which is reserved for future use. It is read-only and its contents are always read as zero. Its address is 17 777 762. It is read on the Internal Data Bus (INTD).

3.4 SYSTEM ID REGISTER

This read-only register [SCCN SYS ID(15:08), SCCM SYS ID(07:00)] contains information uniquely identifying each system. Its address is 17 777 764. It is read on the Internal Data Bus (INTD).

3.5 CPU ERROR REGISTER

The CPU Error Register (Figure 3-1) is a read-only register, consisting of six bits which identify the source of the abort or trap that used the vector at location 4. These bits, which are set when the error occurs, are:

Bit	Name	Function
7	Illegal Halt	Set when trying to execute a HALT instruction when the CPU is in User or Supervisor mode (TMCD ILL HALT).
6	Odd Address Error	Set when a program attempts to do a word reference to an odd address (TMCD ODD ADRS).
5	Non-existent Memory	Set when the CPU attempts to read a word from a location higher than indicated by the System Size register. This does not include Unibus addresses (TMCD CACHE NEXM).
4	Unibus Timeout	Set when there is no response on the Unibus within approximately 10 microseconds (TMCD UBUS TIMEOUT).
3	Yellow Zone Stack Limit	Set when a yellow zone trap occurs (TMCD YEL TRAP).
2	Red Zone Stack Limit	Set when a red zone trap occurs (TMCD SL RED ERR).

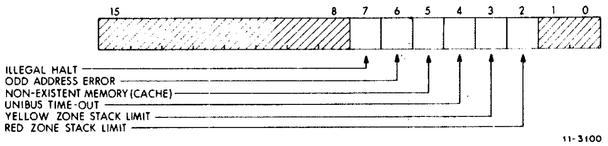


Figure 3-1 CPU Error Register

The CPU Error Register cannot be loaded by the program. It is read via the Internal Data Bus (INTD) at address 17 777 766. The individual bits of this register remain set until they are cleared by a DATO. The several bits of this register are described in Chapter 6.

3.6 MICROPROGRAM BREAK REGISTER (PB)

The Microprogram Break Register (PB) is intended for use as a maintenance tool. When the processor is being operated under the control of the maintenance card, the processor can be halted during any specific microprogram state by loading the address of that state in the PB and setting the switches on the card to the proper positions. A sync point that generates a pulse at T1 (when the microprogram address matches the contents of the PB) is provided on TIGB. During normal operation of the processor, any value can be loaded into the PB without affecting operation of the processor.

The PB is loaded directly from the BR whenever the PB address is generated during an external data transfer; refer to Chapter 5. The PB is an 8-bit register that is loaded from the eight least-significant bits of the BR. When the PB is read, the data must be transferred through the DMX to the BR by a Unibus data transfer operation. The PB is selected by physical address 17 777 770.

The PB [PDRC PB(07:00)] and its use are described in detail in Chapter 4 of this manual.

3.7 PROGRAM INTERRUPT REQUEST REGISTER (PIRQ)

The Programmed Interrupt Request register (PIRQ) allows a program to schedule the execution of various subprograms according to a priority scheme, and at the same time, allowing various levels of hardware interrupt priority to interact with the software priority levels. The register stores interrupt requests set by transferring request data to the PIRQ, and provides information about the requests through encoded data transferred from the PIRQ. Refer to Figure 3-2.

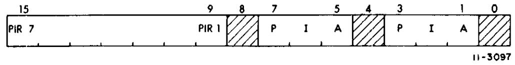


Figure 3-2 Program Interrupt Register

Data is transferred to the PIRQ through the BR whenever the processor recognizes that the physical address is the address assigned to the PIRQ (address 17 777 772). The contents of the PIRQ are then input to the priority arbitration logic of the processor, which uses the information from the PIRQ with information from the Unibus and the PS priority level to determine when requests should be honored.

The data in the PIRQ can be transferred to other devices or to other registers in the processor by addressing the PIRQ during an external data transfer. Because the only outputs from the PIRQ are to the DMX (Unibus Data Multiplexer), all transfers which access the PIRQ are Unibus data transfers. Refer to Chapter 5.

PIRQ [PDRD PIR(15:09)] and PIA [PDRD PIA(02:00)] are described in Chapter 6 of this manual.

3.8 STACK LIMIT REGISTER (SL)

Because the number of locations occupied by a stack is unpredictable, some form of protection against the stack expanding into locations containing other information must be provided. If the processor is operating in Kernel mode, the processor provides for stack overflow detection through the use of the Stack Limit register (SL). Refer to Figure 3-3.

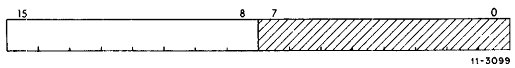


Figure 3-3 Stack Limit Register

The SL is an 8-bit register that is loaded from the eight most-significant bits of the BR whenever the SL is selected by the physical address generated in an external data transfer. This requires address 17 777 775 during a byte transfer, or address 17 777 774 during a word transfer. The data is transferred directly from the BR to the SL; refer to Chapter 5. To read the contents of the SL, however, the SL must be selected by the DMX and the data transferred from the Unibus to the BR. This requires a Unibus data transfer operation. Although the SL and the PB registers share a common DMX input, each register uses a different byte, and only one set is selected at a time. Therefore, when the SL is transmitted on the eight most-significant data lines, all 0s are transmitted on the eight least-significant data lines.

The SL [PDRC SL(07:00)] and the stack limit check operations are described in detail in Chapter 6.

3.9 PROCESSOR STATUS WORD (PS, PSW)

The Processor Status Word [PDRD PS(15:00), Figure 3-4] contains information regarding the processor mode (both current and previous), the register set currently in use, the processor priority, the Trace bit and the Condition Codes. Table 3-1 lists the fields of the PSW. The address of the PS is 17 777 776.

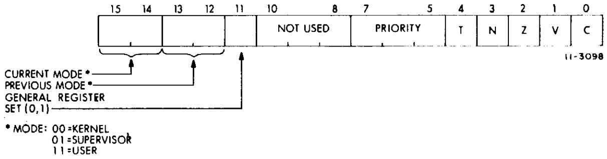


Figure 3-4 Processor Status Word

Refer to drawing PDRD. The PS stores several types of data that are dependent on the process being performed. This data must be stored whenever the processor changes processes; typically, this occurs every time there is an interrupt or a trap. Because the contents of the PS control many parts of the operation of the processor, modifications of the contents are carefully controlled.

Table 3-1
Processor Status Word Bit Assignments

Bit	Name	Utilization
15–14	Current Mode	Specifies the current processor mode as follows: 1. When PS(15:14) = 00, the processor is in Kernel mode; all operations are legal. 2. When PS(15:14) = 01, the processor is in Supervisor mode; HALT, RESET, and SPL instructions are illegal; SUPER address space is used if Memory Management is enabled. 3. PS(15:14) = 10 is an illegal mode; if Memory Management is enabled, a Memory Management abort occurs (refer to Section IV of this manual). 4. When PS(15:14) = 11, the processor is in User mode; HALT, RESET, and SPL instructions are illegal; USER address space is used if Memory Management is enabled.
13–12	Previous Mode	Specifies the processor mode prior to the last trap, interrupt, or loading of the PS.
11	Register Set	Specifies which General Register set is used; if PS11 = 0, register set 0 is selected; if PS11 = 1, register set 1 is used.
10–08	Unused	Unused
07–05	Priority	Sets the processor priority; this priority determines which levels of programmed and external device interrupt requests are honored.
04	Trace	When PS04 = 1, the processor traps to the trace trap vector address after each instruction fetch; this facility is used for debugging programs.

Condition Codes:

03	N	This bit is set when the result of the last data manipulation is negative.
02	Z	This bit is set when the result of the last data manipulation is 0.
01	V	This bit is set when the result of the last data manipulation is incorrect because of an arithmetic overflow.
00	C	This bit is set when a carry occurs during data manipulation.

The four fields of information in the PS are:

1. Processor condition codes

2. Trace (T) bit

3. Processor priority

4. Processor mode control and register set selection bits

Some of the PS bits control the operation of the processor, while others indicate the value of the result of the last data manipulation operation.

In addition to accepting inputs from the BR, the PS receives inputs from the condition-code generation logic. In certain circumstances (the current mode field replaces the previous mode field), some bits of the PS also receive inputs from other bits of the PS. The outputs from the PS during data transfers can be directed to the processor data paths through the BR [by selecting the PS inputs to the internal bus (IBS) and the IBS inputs to the BRMX], or directed to the Unibus through the PS inputs to the Data Multiplexer (DMX). The IBS path is used only for data transfers that implicitly select the PS, such as the stacking operations during interrupt and trap service sequences. When the PS is addressed explicitly, the data is transferred on the Unibus, even if the transfer is to the processor data paths (through the BR).

3.9.1 Reading the PS

1. Implicit reference - The PS word can be gated to the Internal Data Bus by PDRD READ PS H, which is generated by a microprogram IBS field value of 3. This vafue is used in microstates RSD.00, RSD.01, RSD.02, BRK.20, BRK.80, TRP.00, TRP.01, TRP.02, and HLT.00 to get the current PS into the BR. This is shown on the Flows by BR-PS.

2. Explicit reference - The PS word can be read by the program with a reference to address 17 777 776. In this case, the PSW is gated onto the Unibus, from where it is read during a DATI by the processor.

3.9.2 Loading the PS

All used PS bits, with the exception of bit 04, (the T bit) can be written by the program when the PS address (17 777 776) is used (SCCE PS ADRS H is asserted). In this case, the input is BR(15:00) and the clock is a function of MSYN and of UBCB HI BYTE and LO BYTE. These signals are both asserted if the PS is referenced as a word.

1. The Control Codes (bits 03:00) are shown on IRCH and are clocked by UBCB CC DATA.

2. The Priority bits (07:05) are clocked by TMCE CLK LO PS.

3. The Processor Mode bits and the Register Set bit (15:11) are clocked by TMCF CLK HI PS.

The PS may also be loaded under microprogram control (implicit reference). Since the loading logic varies from bit to bit, it is explained with each bit group.

3.9.3 Processor Mode Bits [PS(15:12)]

The current processor mode is stored in PS(15:14) and the processor mode previous to the current one is stored in PS(13:12).

If the current mode is other than Kernel, the HALT, RESET and SPL instructions are illegal: A HALT in Supervisor or User modes causes a trap to 4; RESET or SPL in these modes are NOPs.

When Memory Management is enabled, the mode bits affect PAR/PDR selection, and thus the physical address generated from the virtual address. Refer to Section IV, Memory Management.

3.9.4 Current Processor Mode [PS(15:14)]

The Current Processor Mode bits determine whether certain instructions are allowed or prohibited. The processor mode can be set by moving a data word to the PS at its Unibus address, or through a trap or interrupt service function (which loads a new PS value from the trap or interrupt vector), or through an RTI or RTT instruction (which restores an old PS from the hardware stack). In this last case, PS(15:14) can only be changed to a higher value (i.e., these bits can only be set and not cleared). This allows a Kernel mode program to return to Kernel, Supervisor, or User mode; a Supervisor mode program to return to Supervisor or User mode; and a User mode program only to return to User mode. A User or Supervisor mode program cannot use the RTI instruction to enter the Kernel mode. When a new PS is loaded from the trap or interrupt vector, the old contents of PS15 and PS14 are loaded into PS13 and PS12.

When Memory Management is enabled, the current processor mode selects the mapping for the virtual machine, except for trap and interrupt processing. Supervisor and User programs should not be allowed to change the contents of this field. If the current processor mode is changed, the mapping registers in Memory Management are selected by the set for the new mode. The result of attempting to continue with the same PC value in the new virtual address space is unpredictable.

The entire PS word can be protected from direct transfers by being mapped only into Kernel address space. Refer to Section IV.

PS bits PS15 and PS14 control and indicate the current processor mode. The source of input data is always BR15A and BR14A, whether the PS is loaded by an RTT or RTI instruction, or if a new PS is loaded from a trap or interrupt vector, or explicitly referenced.

3.9.5 Previous Processor Mode [PS(13:12)]

The previous processor mode is used primarily by the MFP and MTP instructions to define which address space to communicate with. During User mode operation, these bits are set to reflect User mode, so that the User program cannot move data into or out of any other address space. During trap or interrupt service, these bits are set to reflect the value contained in the current mode bits prior to the interrupt or trap. In this case, a KERNEL DATI data transfer is used to fetch the new PS value from the vector address; this causes bits 13 and 12 of the PS to be loaded from the old value of bits 15 and 14 instead of from BR(13:12).

During the return from a trap or interrupt service program (via an RTI or RTT instruction), the old PS value is restored from the stacked value. The previous mode bits are protected in the same way as the current mode bits.

3.9.6 PS(15:12) Implicit Write

Refer to Figure 3-5. PS(15:12) can only be set, and not cleared, by their direct-set inputs; they can be both set and cleared when they are clocked. They are clocked only in three machine states (RTI.50, SVC.30 and ZAP.30) when appropriate conditions exist.

When IBS = 2 (LOAD PS) bits 15 - 12 are direct-set if the BSC bits do not require a KERNEL DATI and if the corresponding DATA input is high. These bits cannot be cleared in this manner.

IBS = 2 clocks PS(15:12), thus allowing bits to be cleared, when one of three conditions are present:

1. PS14 = 0, or the mode is Kernel. This is used during RTI and RTT instructions when IBS = 2 in RTI.50.

2. TMCE KERNEL DATI, which is asserted during the service flows (abort, trap and interrupt service, see Chapter 6). IBS = 2 is asserted during SVC.30, when the PS is loaded from the BR.

3. SSRA PS RESTORE is asserted when a Memory Management abort occurs during the service flows. When this happens, the PC and PS of the instruction that caused the abort are restored before servicing the Memory Management abort. In ZAP.30, IBS = 2 and the old PS value is loaded back into the PS.

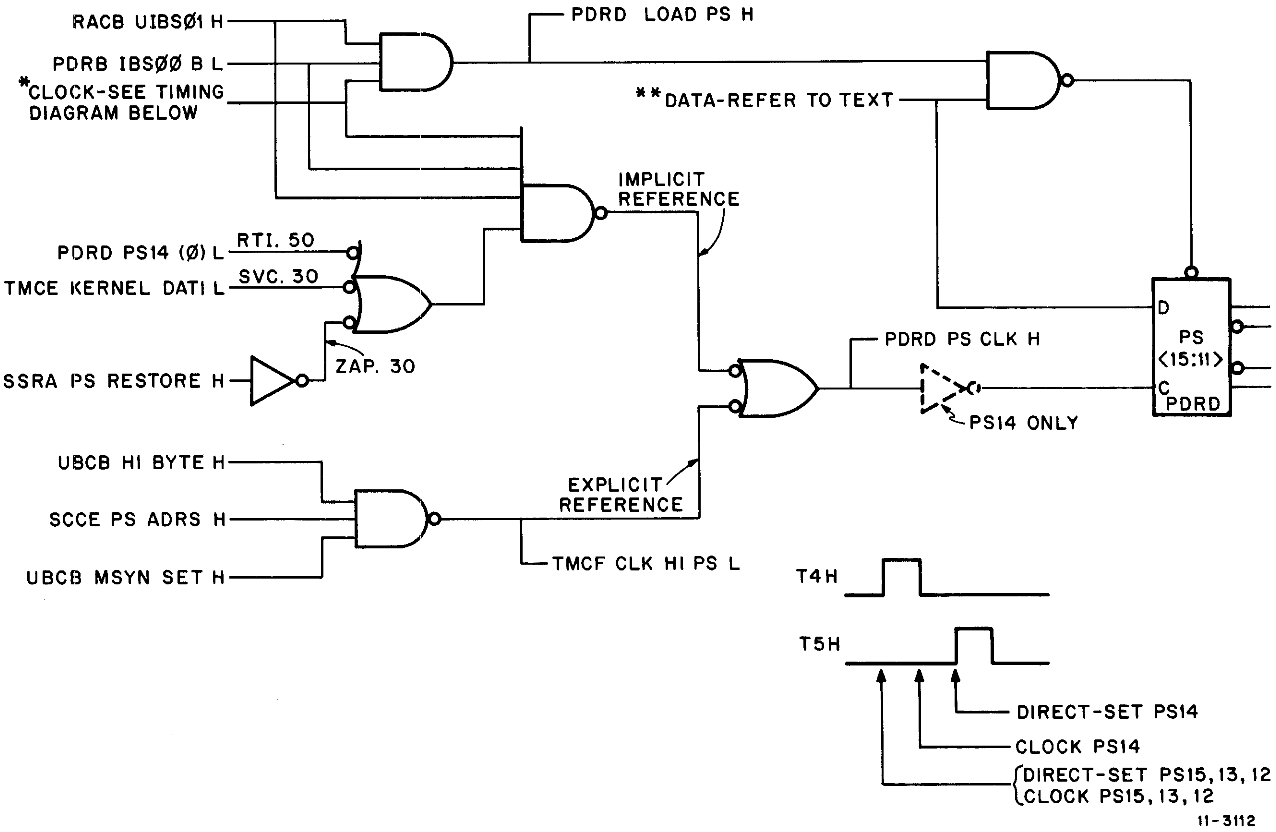


Figure 3-5 PSW Clock and Direct Set Simplified Schematic

Refer to drawing PDRD. Figure 3-5 shows the DATA input to PS(15:12). This input is BR(15:12), except in the case of KERNEL DATI. When KERNEL DATI is asserted, bits 15:14 are clocked from BR(15:14) and bits 13:12 are clocked from PS(15:14). The new processor mode is thus loaded into PS(15:14) and the old processor mode into PS(13:12).

3.9.7 General Register Set Bit (PS11)

PS11 indicates that General Register Set 0 is in use (when cleared), or that General Register Set 1 is in use (when set).

The input to PS11 is BR11A. This bit is loaded in the same manner as PS15 (Paragraph 3.9.6).

3.9.8 Priority [PS(07:05)]

The processor priority is stored in PS(07:05). The 3-bit priority field is interpreted as one of eight priority levels. This level is compared with other requests for control of the system. These requests can be external to the processor, in the case of Unibus requests (BR), or internal, in the case of Program Interrupt Requests (PIR). In general, the purpose of requesting control of the system is to interrupt the current processor program and to run a service routine or higher priority program before returning control to the interrupted program. Refer to Chapter 6 for a description of the priority scheme.

The processor priority level may be set by directly transferring data to the PS, by popping a new PS from the hardware stack, or by loading the PS from an interrupt or trap vector. In addition, the processor priority may be explicitly set by the set priority level (SPL) instruction.

Refer to drawing PDRD. PS(07:05) are clocked in a manner similar to the mode bits (Paragraph 3.9.6), but are not direct-set. The 74S157 multiplexer selects the input: in all cases, except during an SPL instruction, the input is BR(07:05),while during the SPL the input is BR(02:00) A, which corresponds to the position of the new priority bits in the instruction word. TMCE SET PRIORITY H (MSC = 4) controls the multiplexer and gates the clock.

In User or Supervisor modes, the processor priority can only be changed by a transfer to the explicit address of the PS (17 777 776). This is possible only if Memory Management mapping allows it.

3.9.9 Trace Bit (T Bit, PS04)

The Trace (T) bit is provided as a software diagnostic aid. When this bit is set, a processor trap will be vectored through location 14. This trap occurs at the end of the instruction that is being performed when the T bit is being set, unless:

1. The instruction is a Return From Trap (RTT) instruction. In this case, the trap is delayed until the end of the following instruction.

2. The instruction is a Set Priority Level (SPL) instruction. No BRQ STROBE is generated during the execution of an SPL.

3. Some other trap or interrupt condition is honored. In this case, the PS containing the T bit is pushed onto the stack and all Trace operations are deferred until the PS word is popped off the stack at the end of the trap or interrupt service routine.

The T bit cannot be set by moving data to the PS; the only way the T bit can be set is by popping a word off the hardware stack with bit 4 set. This can be done with an RTI, an RTT, or any trap instruction (TRAP, IOT, BPT or EMT), even when the processor is not in Kernel mode. The purpose of inhibiting other methods of loading the T bit is to protect the user from inadvertently setting the T bit while changing the processor priority or condition codes.

The presence of the T bit precludes the use of EXC.80 by E/class*DMO instructions, since the T bit is a trap request. EXC.90 is executed in this case.

3.9.10 Condition Codes

The four least-significant bits of the PS word contain the processor condition codes. These bits store information about the value resulting from any data manipulation during an instruction. The condition codes are not altered to reflect the results of address calculations, but are changed only when an instruction explicitly operates on an explicit unit of data.

The condition codes can also be set to any specific value by transferring a word containing that value to the PS address. The value of the condition codes are altered by every interrupt or trap response function, and by every RTI and RTT instruction. In addition, individual condition-code bits may be manipulated directly, with the condition-code operate instructions. These instructions provide a means to set any one, or more, of the condition codes with a single instruction that requires only one memory reference; a similar set of instructions can clear any one or more bits. The condition codes are used in conditional branch instructions, so the various means of manipulating the condition codes are useful because they permit setting up the PS word to respond in a particular way to various branch instructions.

The logic that senses data conditions and stores the selected indications is on the IRC module and is described in Chapter 1; the gates that control the reading of the condition codes onto the internal data bus are shown on drawing PDRD. When the PS is explicitly addressed at physical address 17 777 776, the data transfer is on the Unibus; the internal bus is used only under direct microprogram control.

The condition codes are loaded automatically with the results of most data manipulations. In addition, the codes can be manipulated by a microcoded instruction that can set or clear individual condition code bits. Any operation that transmits data directly to the processor status word inhibits the setting of the condition codes, because the data transmitted is loaded into PS(03:00) directly. This is done for move instructions that address the PS, RTI instructions that pop a value off the hardware stack into the PS, or interrupt service sequences that load the PS from the interrupt vector.

CHAPTER 4
TIMING GENERATOR

The Timing Generator supplies the clock signals which control the various operations of the KB11-C Processor System. The M8139 module contains all the components of the Timing Generator.

Refer to Figure 4-1. The synchronizer selects one of three clock sources: A 33 MHz crystal clock, an R/C maintenance clock (variable) or a pulse generated by a manual stepper switch. The selected clock signal is routed through a phase splitter/buffer, the output of which consists of two 180° out-of-phase clock signals. These two signals are buffered again and are called TIGC TPB H, TPB L, TF H and TF L. TPB H and TF H are identical and are 180° out of phase with TPB L and TF L, which are also identical.

Separate TPB and TF pulses are provided to separate the timing source required by the TIG module (TPB) from that required by the other modules. A TF failure does not stop the clock.

The TPB pulses drive a five-stage ring counter, the output of which generates gates to generators for time pulses T1 – T5 and for time states TS1 – TS5.

The ring counter is generally stopped during a pulse cycle to allow the data transfer operation in progress to accept the data. It is stopped in T2 for Unibus, Internal Data Bus, interrupt and maintenance operations, and in T5 for Cache operations. The ring counter is also stopped during maintenance operations such as single cycle.

4.1 CLOCK SOURCES

The three sources of timing are the crystal clock, the R/C clock, and the MAINT STPR switch S0 (on the maintenance card). These timing sources are shown on drawing TIGB.

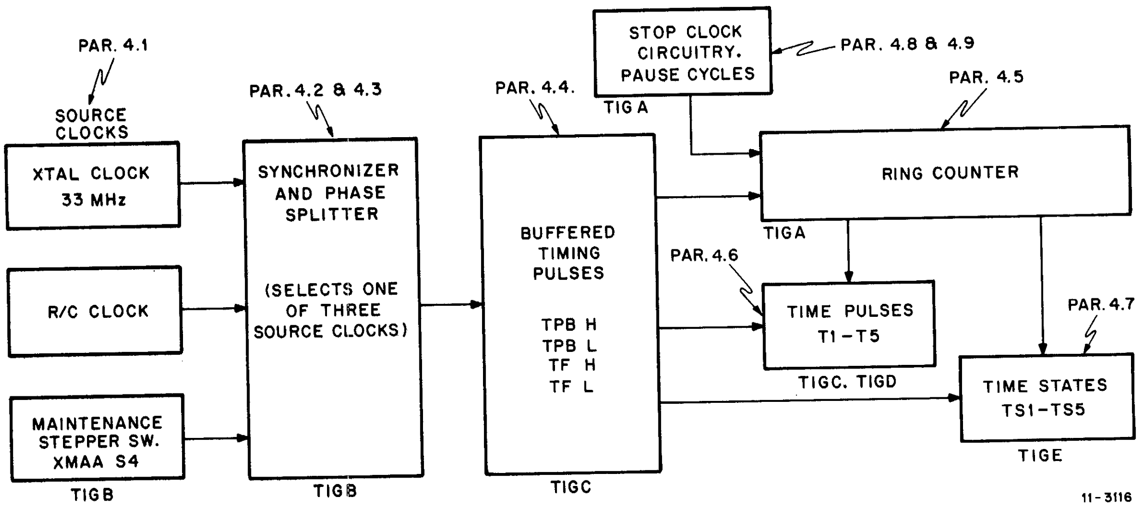


Figure 4-1 Timing Generator Block Diagram

4.1.1 Crystal Clock

The crystal clock provides a constant square wave output of 33 MHz. The oscillator frequency is determined by the LC tuned-collector network and is stabilized by the crystal connected between emitters. The bias network in the base circuits ensures that the oscillator will start when +5 V is applied to the module. The amplified output, TIGB XTAL H, is a +3.5 to 0 V square wave with a 30-ns period.

4.1.2 R/C Clock

The R/C clock is provided for maintenance purposes and can be enabled only when the maintenance card is plugged into the CPU backplane. The frequency of the square wave output, TIGB RC H, can be adjusted as high as 37 MHz by varying potentiometer R104 in the RC feedback network. Thus, the clock pulse period can be narrowed to approximately 27 ns to test for race conditions in the logic.

4.1.3 MAINT STPR Switch

The third source of timing is the manually-operated, single-step MAINT STPR switch S4, located on the maintenance card. This switch is only enabled when maintenance card switches S2 and S3 are both set to 1. Each operation of S4 creates one transition of a given timing pulse. It therefore requires two actuations of S4 to complete a given time pulse.

4.2 SOURCE SYNCHRONIZER

The timing source synchronizer is shown on drawing TIGB. The purpose of the source synchronizer is to select only one timing source at any time and to inhibit the two remaining sources. The synchronizer prevents cycles of improper length and ensures that TIGB SOURCE CLOCK L is in the high (non-asserted) state when switching between sources. Timing source selection is determined by the setting of switches S1, S2, and S3 when the maintenance card is plugged in. If the maintenance card is not installed, the crystal clock is the only source of timing. The following paragraphs describe timing source selection when the maintenance card is plugged in.

4.2.1 Crystal Clock Selection

When maintenance card switch S3 is not set, XMAA S3 L is high. When the RC EN and MS EN flip-flops are not set, the XTAL SYNC flip-flop is set. With maintenance card switches S1 and S2 equal to 0, MS EN will be cleared, as will RC SYNC and RC EN. Therefore, XTAL SYNC is set and the source multiplexer output, TIGB SOURCE CLOCK L, will follow the XTAL H input.

Note that the XTAL EN flip-flop inhibits the maintenance module switch S3 inputs to the RC EN flip-flop. Therefore, the XTAL SYNC flip-flop must be cleared before a timing source change can be accomplished. The RC EN and MS EN gating input to the XTAL SYNC flip-flop ensures that these sources have been disabled before XTAL EN is allowed to gate the XTAL H pulse through the source multiplexer.

4.2.2 RC Clock Selection

The RC clock is selected as the timing source when maintenance card CLK switch S3 is on RC, and S2 and S1 are both set to 0. When the XMAA S3 L input is low, the RC SYNC flip-flop will be set. As a result, the RC EN flip-flop will be set and the source multiplexer output, TIGB SOURCE CLOCK L, will then follow the TIGB RC H input. TIGB XTAL EN (0) H and TIGB MS EN (0) H are fed back to inhibit TIGB RC SYNC D inputs to ensure that the enable flip-flops are cleared before the timing source can be changed.

4.2.3 MAINT STPR Selection

The maintenance card S2 and S1 switches are both set to 1 to allow single timing pulses to be generated by MAINT STPR switch S4. The XMAA S1 L and XMAA S2 L inputs are both low. The resultant input to the MS EN flip-flop D input causes the flip-flop to be set. On the following TIGB XTAL H and TIGB RC H clock pulses, the XTAL SYNC and RC SYNC flip-flops will be reset. Succeeding clock pulses will then reset the XTAL EN and RC EN flip-flops. MS EN (1) H is ANDed with STEP (1) H to assert the TIGB SOURCE CLOCK L output of the source multiplexer. Each time the MAINT STPR switch S4 is operated, the STEP flip-flop toggles. The MAINT STPR switch must be actuated twice to complete a single TIGB SOURCE LOCK L output pulse. Removing the S2 or S1 input conditions the MS EN flip-flop to be cleared. MS EN (0) L direct-clears STEP to condition it for the next time the ING TP function is selected.

4.2.4 Synchronization

A feature of the source synchronizer is that the output level is maintained high (non-asserted) while the timing source is being changed. The timing diagram in Figure 4-2 shows the TIGB SOURCE CLOCK L output as the maintenance card CLK switch is changed from XTAL to RC. With the XMAA S3 L input low (RC clock selected), the XTAL SYNC flip-flop is cleared on the next TIGB XTAL L clock pulse going low.

One XTAL H clock pulse later, XTAL EN will be cleared, enabling the D input to the RC SYNC flip-flop. The next time TIGB RC H goes low, RC SYNC will be set. The difference in XTAL H and RC H pulse widths is exaggerated in Figure 4-2 to indicate that the clock pulses are completely independent.

Note that the SYNC and EN flip-flops are clocked on the trailing edge of the source locks so that the gating level to the source multiplexer is always removed as the clock input is non-asserted. This provides a clean leading edge for TIGB SOURCE CLOCK L. Note also that only half a clock period is available for the enable flip-flop to change states and gate the associated clock source through the multiplexer.

4.3 PHASE SPLITTER/BUFFER

The Phase Splitter/Buffer, shown on drawing TIGB, is driven by TIGB SOURCE CLOCK L from the source synchronizer to produce timing pulse outputs TIGB CLOCK L and TIGB CLOCK H. The TIGB CLOCK L output pulses are in phase with TIGB SOURCE CLOCK L.

4.3.1 Level Converter

Transistors Q65 and Q66 convert the TIB SOURCE CLOCK L output to the level required at the phase splitter inputs. A low logic input at the base of Q65 causes this transistor to conduct, thus grounding the common emitter of Q65 and Q66. The +V2 reference voltage applied at the base of Q66 cuts this transistor off, causing no current to flow through Q66 and R122. Thus, a low input provides a low output. When TIGB SOURCE CLOCK L goes high, Q65 cuts off, and the +V2 reference at the base of Q66 allows current to flow through Q66 and R122 to provide a high output.

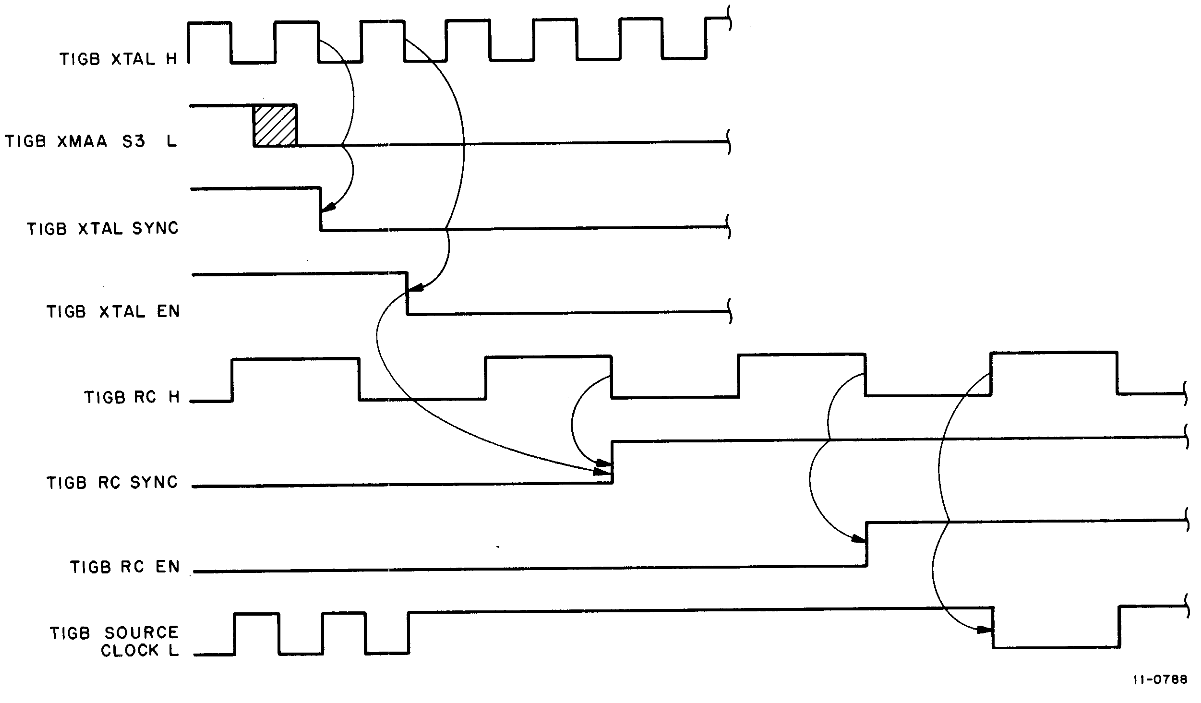


Figure 4-2 Timing Source Synchronization

4.3.2 Phase Splitter

The phase splitter consists of two emitter-coupled 2N3009 transistors, Q61 and Q62. When TIGB SOURCE CLOCK L is not asserted (high), Q61 turns on. A fixed bias at the Q62 base holds that transistor cut off. Under these conditions, the TIGB CLOCK H output provided by buffer Q53 and Q54 is low because Q61 is conducting. Q54 is on.

When TIGB SOURCE CLOCK L starts to go low, as the result of a clock pulse, the base of Q61 goes negative with respect to the Q62 base. More current flows through Q62, causing a greater voltage drop across the Q62 collector resistor, R109-R111. Less voltage is developed across common emitter resistors R89-R96, increasing the forward bias on Q62. As a result, when Q62 starts to conduct more current, Q61 starts to cut off. This circuit is a differential amplifier that responds to slight changes of the input signal at high speed. When TIGB SOURCE CLOCK L starts to go positive, Q61 turns on and Q62 cuts off in the same manner. The switching action of Q61 and Q62 follows the TIGB SOURCE CLOCK L signal with about a 1 ns difference between TIGB CLOCK H and TIGB CLOCK L.

4.3.3 Buffers

Each buffer stage consists of a 2N3009 and a 2N4258 transistor. When Q61 turns off as a result of a low source synchronizer output, Q53 is turned on and Q54 is cut off. Thus, the TIGB CLOCK H output goes high, 180° out-of-phase with the TIGB SOURCE CLOCK L input. At the same time, Q62 turns on and the positive collector cuts off Q5 and forward-biases Q56. Therefore, TIGB CLOCK L goes low in phase with the TIGB SOURCE CLOCK L input from the source synchronizer.

4.4 TIGC TPB AND TF

The outputs of the Phase Splitter/Buffer, TIGB CLOCK H and CLOCK L, are buffered to generate the Time Pulses Buffered, TPB H and TPB L and the Free Clock pulses, TF H and TF L. TPB H and TF H are in phase with TIGB CLOCK H, and are the complement of TPB L and TF L.

The TF pulses are used throughout the **KB11-C** for synchronization. The TPB pulses are used only on the M8139 module.

TPB H and TF H are driven from CLOCK H; TPB L and TF L are driven from CLOCK L. With this exception, the circuits that generate these four pulses are identical: when TIGB CLOCK is high, the NPN transistors conduct, the PNPs are cut off, and the output is high. When TIGB CLOCK is low, the NPNs are cut off, the PNPs conduct, and the output is low.

4.5 RING COUNTER

The Ring Counter is shown on drawing TIGA. It consists of the two edge-triggered D flip-flops, T1 and T1A, of the six J-K flip-flops T2 - T5, T2A and T5A, and their associated circuitry. Refer to Figure 4-7 for the description that follows.

Start-Up and Normal Cycle

The ring counter is cleared by ROM INIT, which is asserted on power-up, power-down, and when the Console START switch is depressed while the HALT/ENABLE switch is in the HALT position. T4 is not cleared by ROM INIT directly, but by a flip-flop that is set by ROM INIT. When ROM INIT is negated, the trailing edge of the next TPB L clears this flip-flop.

When the ring counter has been cleared, the J input of T4 is high [T5 (0) H, T2 (0) H and T1A (0) H are all high] and the next TPB sets T4.

It should be noted that the D flip-flops (T1 and T1A) are clocked by the trailing edge of TPB L, while the J-Ks are clocked by the trailing edge of TPB H. Both of these trailing edges occur at the same time.

The next TPB after the one that sets T4, sets T5 and T4 complements (both J and K high) and is reset. If TIGA STOP T1 L is high, the next TPB complements T5 (resets it) and sets T1. T5 (0) H is now high and asserts STOP T1 L. The TPB that follows clears T1 and sets T2 but, since the common input to T2-K and T3-J is low at this time [due to T2 (0) H] T3 is not set. T2 (0) H is now low, and the next TPB toggles T2 (clears it) and sets T3. T5 (0) H, T2 (0) H and T1A (0) H are all high, thus allowing T4 to be set as T3 toggles.

The ring counter flip-flops are used to gate the timing pulses T1 – T5. Note that there are two T2 and two T5 flip-flops. In both cases, the second flip-flop (T2A and T5A) is used to generate its corresponding timing pulse. These flip-flops are used to prevent the generation of more than one timing pulse (T2 or T5) during Pause cycles: T2A and T5A are reset by the TPB H following the one that sets them, while T2 and T5 remain on for the duration of the Pause.

4.6 TIMING PULSES, T1-T5

The switching times of the flip-flops used in the ring counter are not very precise; therefore, thė flip-flop states are not used directly for processor timing. Instead, high-speed transistors are used to generate the timing pulses. The timing pulse generator schematics are shown on drawing TIGC and TIGD.

Each of the timing pulse generators gates the Phase Splitter/Buffer clock output, TIGB CLOCK H or L, with a ring counter output to generate the timing pulse associated with that state. Figure 4-3 shows how T5A (1) H and T5A (1) L are gated with CLOCK H and CLOCK L to provide the T5 H and T5 L timing pulses.

Note on drawing TIGC that the TIGB CLOCK H and L signals are carried by two separate lines to the timing pulse drivers; these lines are terminated at the TIGD T5 L circuits by diode terminators and at T5 H by a 33 ohm resistor to ground. These lines are transmission lines, designed to guarantee the integrity of the CLOCK H and CLOCK L signals from the phase splitter to the intended pulse generator.

The +V and -V voltages shown on the schematics are taken from diode dividers shown on TIGB for +V5 to +V1 and on TIGE for V3 to -V1.

Since the circuits for T(1:5) H are identical, as are those for T(1:5) L, only the T5 schematics are explained below.

Figures 4-4 and 4-5 are simplified schematics of TIGD T5 H and L, respectively. Q53 and Q54 on the first figure and Q55 and Q56 on the second, are the output of the Phase Splitter/Buffer, TIGB CLOCK H and L. The diode terminators are selected to produce a pulse amplitude of approximately 0 to +3.0 V for T5 H and of approximately +3.0 to 0 V for T5 L. Q32 and Q50 are not shown on Figures 4-4 and 4-5. These transistors are turned off when TIGA T5A is asserted, thus allowing Q31 and Q49 to conduct. Q32 and Q50 conduct when TIGA T5A is negated and turn Q31 and Q49 off.

NOTE

The voltages shown on Figures 4-4 and 4-5 are approximate. They are based on a diode voltage drop of 0.7 V.

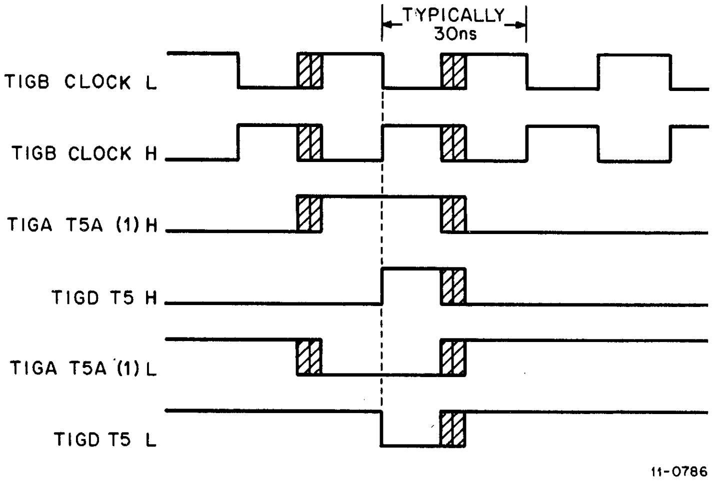


Figure 4-3 Timing Pulse Generation

4.6.1 T5 H

Refer to TIGD and to Figure 4-4(a). The output transistor pair, Q1 and Q2, is arranged to give a push-pull type output. Diode D2 between the two bases, along with the resistor network consisting of R12 (15K to +15 V) and R10 (3K to -15 V), biases the transistor pair Q1 and Q2 so that a small voltage change at the base input turns one transistor on and the other off. This arrangement has the effect of reducing the propagation time from the CLOCK H signal to the output time pulse TIGD T5 H. Diode D1 clamps the bias at a level such that T5H is at approximately 0 V when either CLOCK H is low or the gate transistor Q31 is off. Diode D3 prevents the bias circuit from saturating Q2 by clamping the signal to +V5, or approximately 4 V.

When TIGA T5A (1) H is low, Q32 conducts and Q31 is cut off. When T5A (1) H goes high, Q32 cuts off. Q31 cannot turn on at this time, since its emitter is negative (CLOCK H at approximately 0 V, determined by the base voltage of Q54) with respect to its base (-15 V - D36 to +V2 = approximately +0.7 V). The voltage at the base of Q1 is approximately -0.7 V and that at the base of Q2 is one diode drop more positive, Q2 is off and Q1 conducts; T5 H is low.

Figure 4-4(b) shows the circuit when TIGB CLOCK H goes high. Q54 is now off and Q53 is on. The emitter of Q54 is now positive with respect to its base and it conducts. The voltage at the base of Q1 and Q2 becomes more positive; Q1 conducts and Q2 is turned off. T5 H goes high.

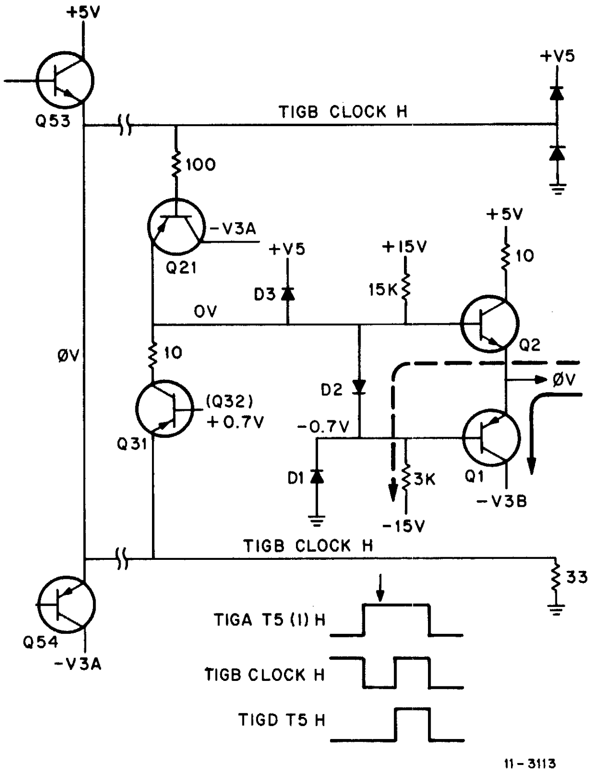


Figure 4-4a

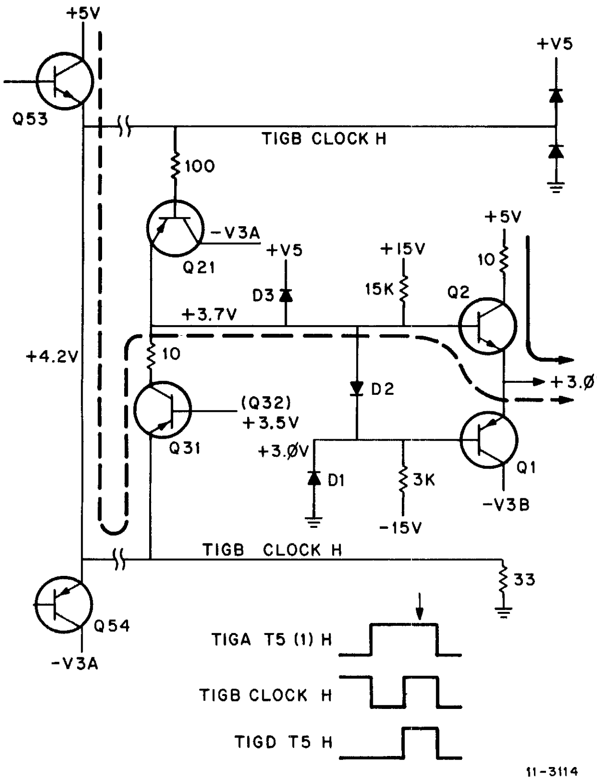


Figure 4-4b

Figure 4-4(c) shows the end of the T5 H pulse. TIGB CLOCK H goes low: Q53 turns off and Q54 turns on. TIGA T5A (1) H goes low and turns Q31 off. Q21 turns on and the voltage at the base of Q1 and Q2 goes negative, turning Q2 off and Q1 on, thus making TIGD T5 H low. Q21 speeds this transition by providing a discharge path for the charge left in the base bias circuit.

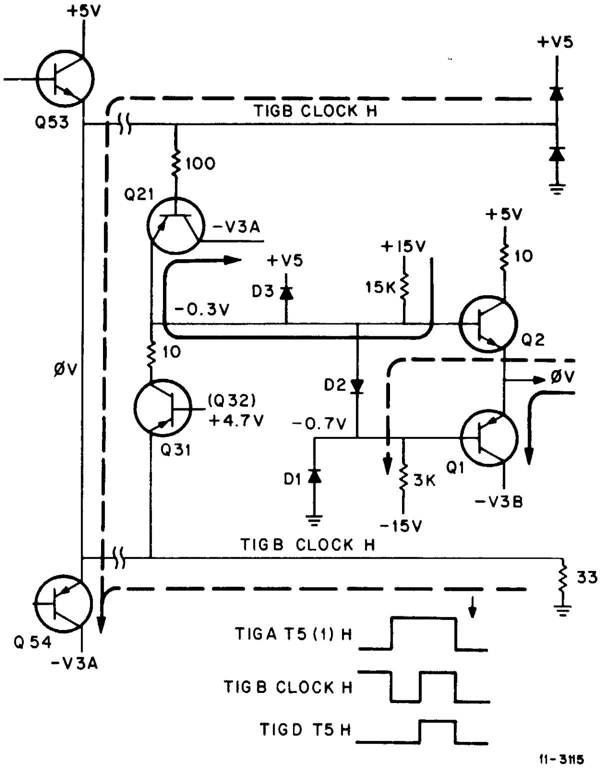


Figure 4-4c

4.6.2 T5 L

Refer to TIGD and to Figure 4-5(a). The output transistor pair, Q19 and Q20, is arranged to give a push-pull type output. Diode D29 between the two bases, along with the resistor network consisting of R61 (1K to +15 V) and R10 (4.7K to -15 V), biases the transistor pair Q19 and Q20 so that a small voltage change at the base input turns one transistor on and the other off. This arrangement has the effect of reducing the propagation time from the CLOCK L signal to the output time pulse TIGD T5 L. Diode D30 clamps the bias at a level such that T5 L is at approximately +3 V when either CLOCK L is high or the gate transistor Q49 is off. Diode D28 prevents the bias circuit from saturating Q19 by clamping the signal to ground.

When TIGA T5 (1) L is high Q50 conducts and Q49 is cut off. When T5 (1) L goes low Q50 cuts off. Q49 cannot turn on at this time, since its emitter is positive (CLOCK L at approximately +3.5 V, determined by the base voltage of Q55) with respect to its base (15 V - D49 to +V3 = approximately +2.8 V). The voltage at the base of Q20 is approximately +3.5 V and that at the base of Q19 is one diode drop more negative. Q20 is conducting and Q19 is off; T5 L is high.

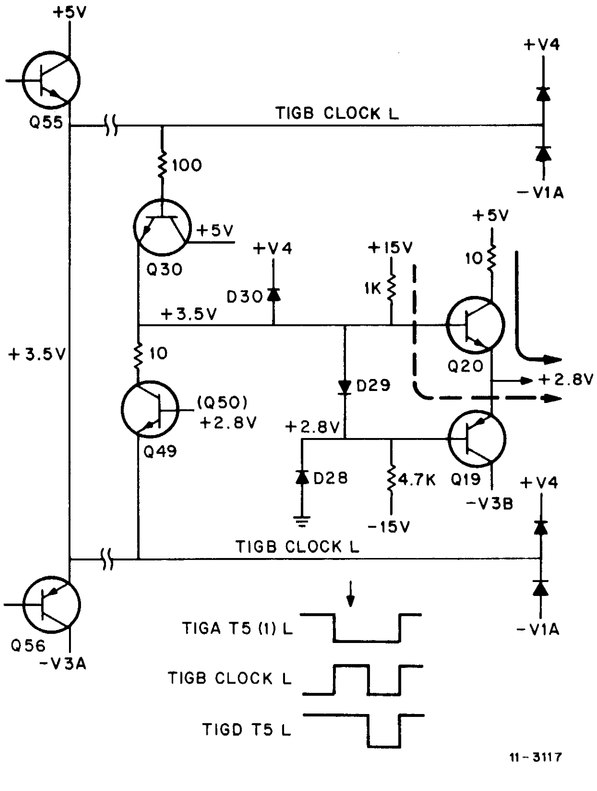


Figure 4-5a

Figure 4-5(b) shows the circuit when TIGB CLOCK L goes low. Q55 is now off and Q56 is on. the emitter of Q49 is now negative with respect to its base and it conducts. The voltage at the base of Q20 and Q19 becomes more negative; Q19 conducts and Q20 is turned off. T5 L goes low.

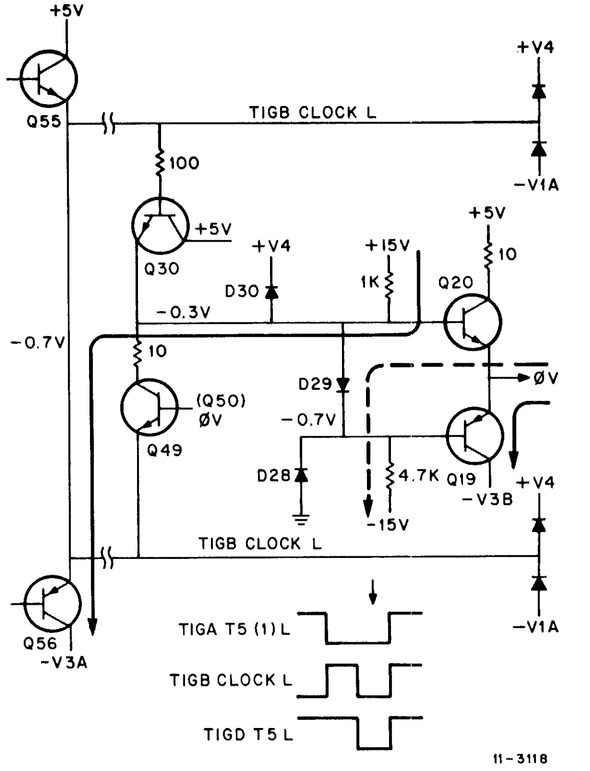


Figure 4-5b

Figure 4-5(c) shows the end of the T5 L pulse. TIGB CLOCK L goes high; Q56 turns off and Q55 turns on. TIGA T5A 1 L goes high and turns Q49 off. Q30 turns on and the voltage at the base of Q19 and Q20 goes positive, thus making TIGD T5 L high. Q30 speeds this transition by providing a discharge path for the charge left in the base bias circuit.

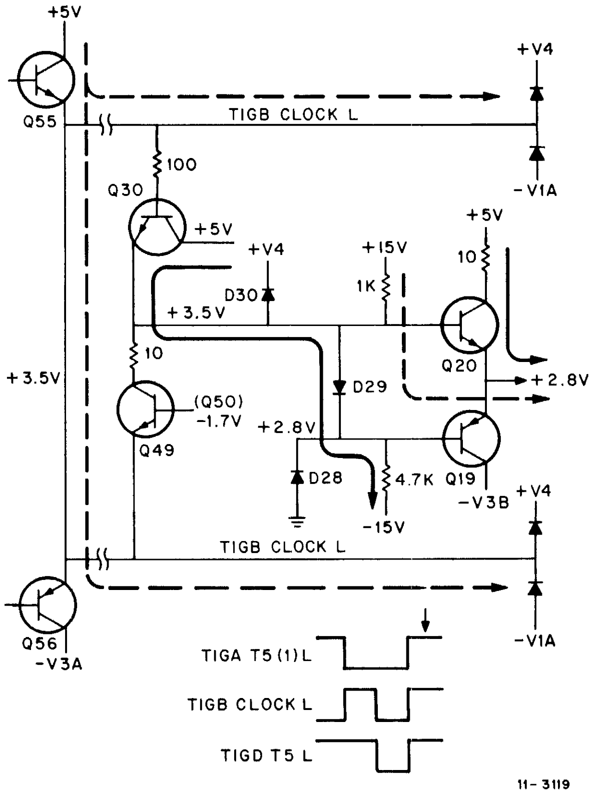


Figure 4-5c

4.7 TIME STATES (TIGE TS1 L–TS5 L)

Refer to Figure 4-6. The Time State pulses, TIGE TS1 L through TS5 L are generated from the ring counter flip-flops and TIGB TPB H. The leading edge of these pulses corresponds to that of the timing pulse of the same number (e.g., TS1 to T1)

The time states are used throughout the **KB11-C** and are on for two time pulse durations (e.g., TS1 is on from the leading edge of T1 to the leading edge of T3).

These time state pulses are provided for use in areas where timing is not critical, in order to reduce the load requirement of the timing pulses.

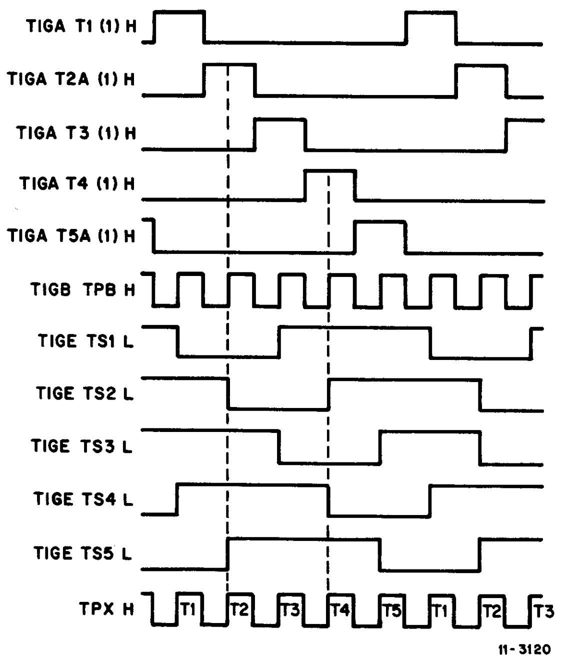


Figure 4-6 Time States

4.8 PAUSE CYCLES AND CLOCK BR

The ring counter is stopped during Pause cycles, except in the case of a Cache read hit cycle. The stop occurs during T5 for Cache Pause cycles and during T2 for Unibus, Interrupt (INTR) or Internal Data Bus (INT D) cycles. The INTR Pause cycle is one where UBSD=1; for all other Pauses, UBSD=2 or 3 [TIGA PAUSE H=ROM 40 asserted (UBSD01=1)].

Table 4-1 is a summary of Stop and Pause conditions.

4.8.1 Synchronous Pauses

4.8.1.1 Internal Bus (INT D) Pause (T2) - Refer to Figure 4-7. During a Pause for an INT D read, a 90-ns delay is inserted between T2 and T3 by the S0 and S1 flip-flops.

The ring counter is stopped by the low output of the 74S65 gates which cause a low input to the K input of the T2 flip-flop. The flip-flop cannot be reset until this input becomes high. The low is caused by the two gates that have SAPN NOT CACHE ADRS H as inputs. Since the INT D registers have Unibus addresses, this signal is high. S1 (0) H and S0 (0) H are also high, as well as TIGA PAUSE H (UBSD=2 or 3, Bus Pause). S1 and S0 are clocked by TPB H and count up to 3. At this time, both S1 (0) H and S0 (0) H are low, the output of the 74S65 gates goes high, and on the net TPB pulse, T2 (1) H is cleared, T3 (1) H is set, and the ring counter is restarted.

4.8.1.2 Cache Pause (5) - Refer to Figure 4-7. The ring counter stops in T5 during a Pause for a Cache cycle. A read hit Cache Pause cycle is the only Pause cycle in which the ring counter is generally not stopped; all other Cache cycles stop the counter. CCBC MEMSYNC H is asserted by the Cache when it has completed a memory cycle.

TMCF CACHE ADRS H is asserted during a pause for a Cache cycle and, in conjunction with TIGA PAUSE; if there is no abort pending, and if TIGA MEMSYNC is not asserted, then TIGA STOP T1 L is asserted and prevents T1 from being set until CCBC MEMSYNC H is asserted by the Cache. When this occurs, TIGA MEMSYNC (0) H goes low. The next TPB sets T1, clears T5 and restarts the ring counter.

4.8.2 Asynchronous Pauses

Synchronizing flip-flops are required during asynchronous Pause cycles in order to minimize the possible instability of flip-flops when clocked at the same time that their data input is changing.

4.8.2.1 Unibus Pause (T2) - Refer to Figure 4-7. During a Unibus Pause cycle, the ring counter is stopped during T2, as for the INT D Pause.

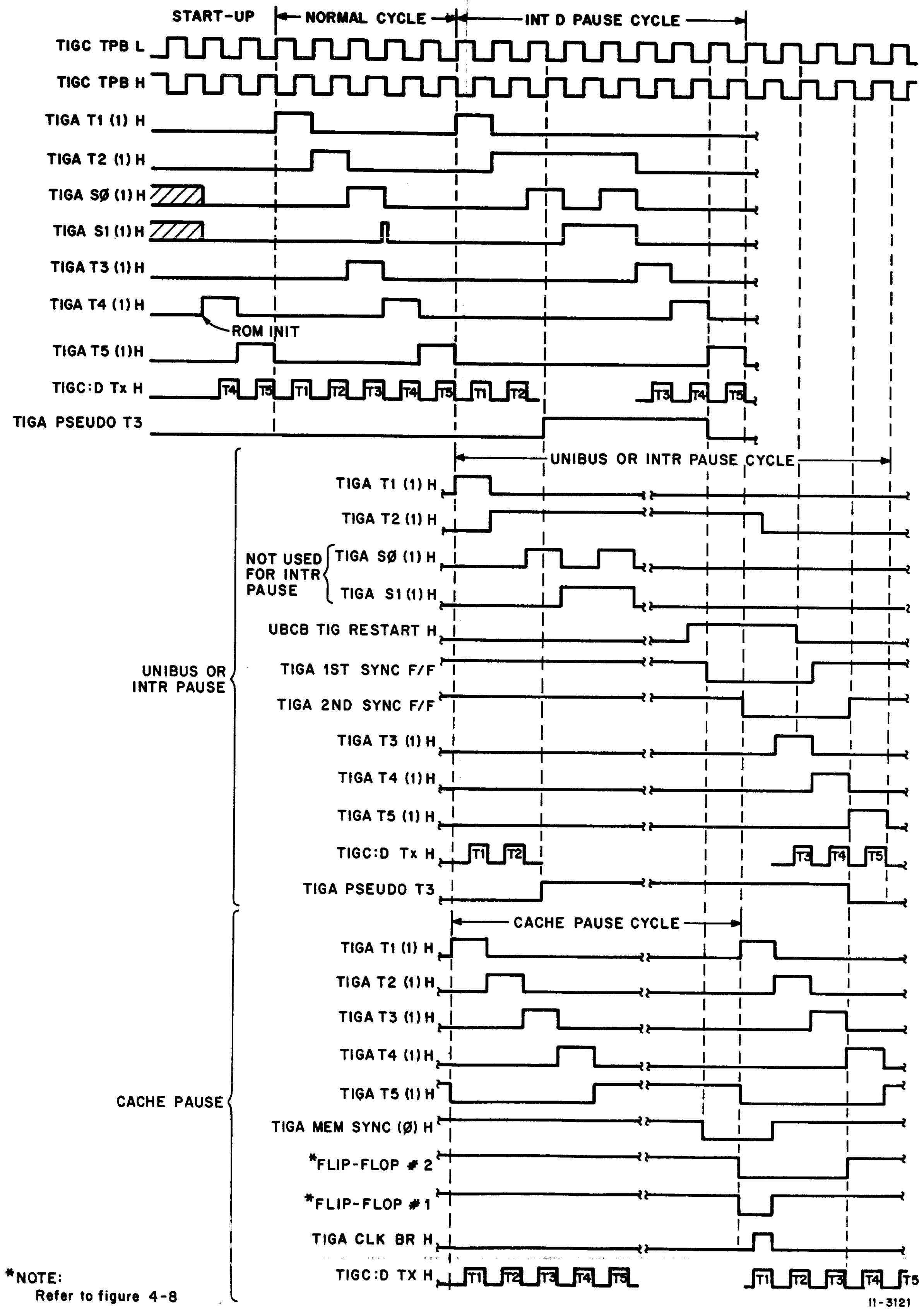
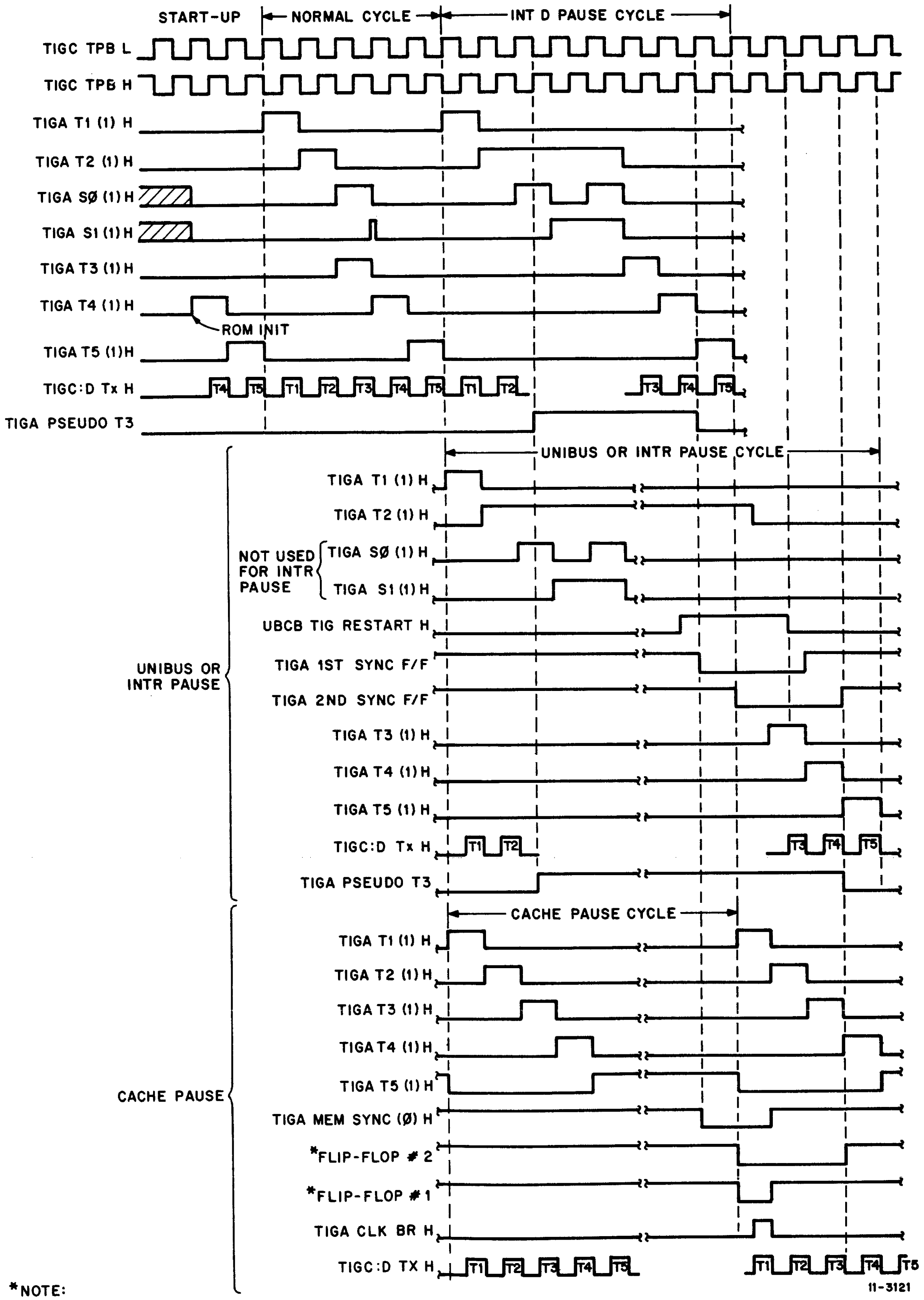


Figure 4-7 Timing Generator and Pauses
(Figure repeated on next page)



*NOTE:
Refer to figure 4-8

Figure 4-7 Timing Generator and Pauses

Table 4-1
Ring Counter Stop and Pause Conditions

	STOP IN T2	
Internal Bus Pause	Stop:	SAPN NOT CACHE ADRS H TIGA PAUSE H (UBSD = 2 or 3) TIGA S0 (0) H or TIGA S1 (0) H
	Restart:	S0 and S1 count to 3 (90 ns).
Unibus Pause CPU Control Registers	Stop:	Same as Internal Bus Pause
	Restart:	Same as Internal Bus AND UBCB TIG RESTART H (BUS SSYN)
Interrupt Pause	Stop:	UBSD = 1 (INTR Pause) UBCD EXT BRQ H
	Restart:	UBCB TIG RESTART H (Passive Release or BUS INTR)
Single ROM Cycle	Stop:	TIGB ROM+UPB (1) H
	Restart:	CONTINUE or MAINTENANCE (XMAA S4) switches
	STOP IN T5	
Cache Pause	Stop:	TMCF CACHE ADRS H TIGA PAUSE H (UBSD = 2 or 3) No Aborts (not TMCC ABORT H)
	Restart:	TIGA MEMSYNC (1) H
Single Bus Cycle	Stop:	TIGB SINGLE CY L TIGA PAUSE H
	Restart:	CONTINUE or MAINTENANCE (XMAA S4) switches

The Unibus Pause is started by the same two gates that start the INT D Pause, in addition to the gate that has UBCA UNIBUS ADRS H as an input. SCCD INTD REG (1) L is high, since the address does not refer to an Internal Bus register.

There are two synchronizing flip-flops for this gate: the first rank flip-flop is the one that has UBCB TIG RESTART L as its input; the second rank flip-flop has the output of the first rank flip-flop as its input. The output of the second rank synchronizing flip-flop is high at this time. The output of the 74S65 gates is low, T2 (1) H is not cleared, and T3 (1) H is not set. The S0 and S1 flip-flops count to 3, at which time the NOT CACHE ADRS gates are disabled. When BUS SSYN is received, UBCB TIG RESTART is asserted. The first rank synchronizing flip-flop is set by the next TPB L, and the second rank flip-flop by the TPB after that. This disables the UNIBUS ADRS gate, and the output of the 74S65S goes high, allowing the ring counter to restart.

When reading the Control Registers (PS, SL, PIR, PIA and PB - see Chapter 2, Paragraph 2.3.2) SSYN is generated by the processor; in this case the 90 ns S0-S1 delay and the synchronizing flip-flop delays may be concurrent.

4.8.2.2 INTR Pause (T2) - Refer to Figure 4-7. The interrupt (INTR) Pause cycle is similar to the Unibus Pause cycle.

The ring counter is stopped in T2 by the UBCD EXT BRQ H gate on the lower 74S65. This gate is asserted during an INTR Pause cycle (UBSD= 1); the output of the second rank synchronizing flip-flop is high at this time. UBCD EXT BRQ H is asserted when any of TMCA HONOR BR(4:7) are asserted. The T2 flip-flop remains set and the T3 flip-flop cleared until the second rank synchronizing flip-flop is set. S1 and S0 count up but have no effect, since they are ANDed with TIGA PAUSE H (UBSD= 2 or 3), which is low. UBCB TIG RESTART H is asserted either by the receipt of INTR or by a passive release of the Unibus (UBCA PASSIVE L). The first rank synchronizing flip-flop is set by the next TPB L, and the second rank flip-flop by the TPB after that. This disables the EXT BRQ gate, and the output of the 74S65 goes high, allowing the ring counter to restart.

4.8.3 CLK BR, BRA

During any cycle during which UBRK (load BR) is asserted, the BR is loaded at the proper time. During a Cache Pause cycle, the data is loaded into the BR at MEMSYNC+30 ns. During any other type of cycle, the BR is loaded at T5+30 ns. These operations are independent of when T1 occurs.

4.8.3.1 Non-Cache Cycles - Refer to Figures 4-8 and 4-9. TMCF CACHE ADRS H is asserted during all Cache cycles. RACB ROM 40 L is asserted and TIGA PAUSE H is high during all Bus Pause cycles (UBSD=2 or 3). When either CACHE ADRS or PAUSE are low, gate 2 is high, T5A(1) is gated through gate 3 and the OR gate and sets flip-flop 1 one clock period later. The following TPB L, which occurs at T1, asserts TIGA CLK BR (and BRA) if RACA UBRK H is asserted.

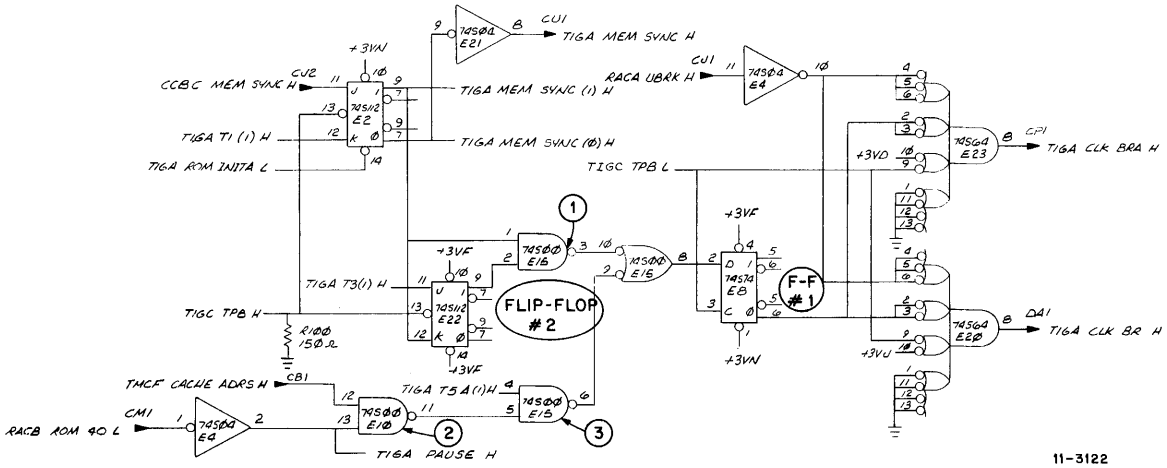


Figure 4-8 Clock BR Circuit (Part of D-CS-M8139-0-1, Sheet 3)

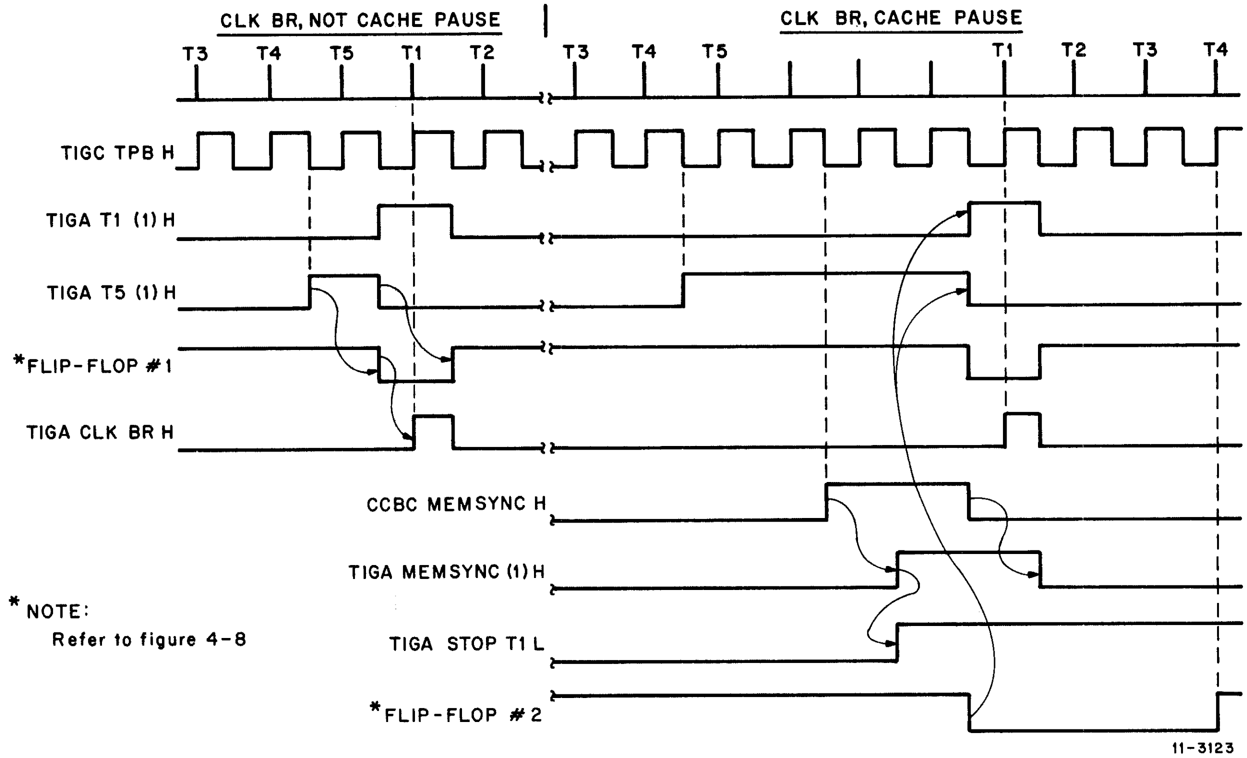


Figure 4-9 Clock BR Timing

4.8.3.2 Cache Cycles - Refer to Figures 4-8 and 4-9. MEMSYNC gates the data from the Cache into the BR during a Cache DATI or DATIP. Flip-flop 2 is set prior to the Pause cycle by T3.

Upon entering a Cache Pause cycle, gate 1 is enabled. When CCBC MEMSYNC H causes TIGA MEMSYNC to set, the output of gate 1 goes low, the output of the OR gate goes high, and flip-flop 1 is set at the same time as T1 (1) H (the ring counter is restarted by TIGA MEMSYNC). Since RACA UBRK is asserted, CLK BR is asserted 15 ns later by TPB L, which occurs at the same time as T1.

Flip-flop 1 is on for only one clock period to ensure that only one BR clock pulse is generated.

4.9 MAINTENANCE STOPS

4.9.1 Single Cycle Mode

When the processor is halted and placed in the S BUS CYCLE mode of operation from the console, the TIGA SNGCY flip-flop is direct-set to assert TIGA STOP T1 and cause the processor to halt after each single bus cycle is completed (TIGA PAUSE). When the CONT switch is pressed, TIGB CONT is asserted and clocks the J-K flip-flop that sets TIGA CONT (1) on the next TIGB TPB pulse going high. This enables the K input to the SNGCY flip-flop so it will reset on the next TPB pulse going high.

The processor enters T1 and proceeds through another bus cycle. As soon as T1 is entered, the flip-flop controlled by the CONT switch is reset. The CONT flip-flop resets on the next clock pulse and the SNGCY flip-flop is again set on the trailing edge of that clock pulse. As a result, STOP T1 is again asserted to stop the processor after a single bus cycle.

Since TIGA CLK BR is generated by either MEMSYNC or T5A (1), independently of T1, the data is loaded into the BR 30 ns after T5. During Single Cycle, the clock is stopped in T5, and the data from the current cycle could not be displayed if the BR were clocked by T1 (after the clock has been restarted).

4.9.2 ROM+UPB

SINGLE ROM CYCLE operation (S1=0, S2=1) stops the clock in T5 of every ROM cycle.

The UPB STOP (S1=1, S2=0) operation stops the clock in T5 when PDRC PB COMP H is high. This signal is asserted when the microprogram ROM address equals the contents of the Program Break Register [PDRC PB(07:00)]. This read/write register is accessed at address 17 777 770.

Maintenance module switch inputs XMAA S1 and S2 are decoded, ORed and input to the TIGB ROM+UPB (1) H flip-flop, which is cleared by T5 (1) L and clocked by the following TPB L (at the trailing edge of T1 (1) H). Since the CONT flip-flop is cleared, the clock is stopped in T2.

4.9.3 TIGB CONT L

It should be noteu that, except for single clock cycle operation, either the Console CONT switch or the maintenance stepper XMAA S4 can be used. XMAA S4 must be used for single clock cycling.

CHAPTER 5
DATA TRANSFERS

This chapter examines the types of processor data transfers (Paragraph 5.1), discusses the Unibus interface, in general terms (Paragraph 5.2), and describes processor data exchange with the Unibus (Paragraph 5.3).

In order to execute instructions, the processor exchanges data with the Cache and with Unibus devices; it contains the Unibus arbitrator, which decides which device obtains the use of the Data Section of the Unibus. The Unibus arbitrator is a part of the processor priority network, which is described in Chapter 6.

In order to exchange data with either the Cache or with a Unibus device, the processor must supply the following information:

1. An *Address*, which defines the device or the location in memory with which the data exchange is to take place; address generation is described in Section IV of this manual.

2. *Control* information, which specifies the direction of the data transfer; the C bits determine the type of transfer and are described in this chapter.

3. *Data*, in the case of a transfer from the processor to the Cache or to the Unibus; data is supplied to the Cache by the BR and to the Unibus by the Data Multiplexer (DMX), both of which are described in Chapter 2.

5.1 PROCESSOR DATA TRANSFERS

The processor requires two ROM states to execute a data transfer; a BUST (BUs STart) cycle and a Bus Pause cycle, during which the transfer of data takes place. A BEND (Bus END) cycle may replace the Pause cycle if the transaction is not to be completed (either due to error or to the microprogram). Stack and Address errors (aborts, refer to Chapter 6) are detected prior to the completion of a Bus cycle and cause a BEND. Conditions in the microprogram which can cause a BEND are those where Bus cycles are started in anticipation of certain forks or branches. If the fork or branch results in a condition which does not require the Bus cycle to be completed, it is stopped by a BEND. An example of this is found on Flows 5: D12.00, D12.80 and D12.90 all do a BUST and branch to one of three cycles; one of these, D12.70, does not require a Bus cycle and does a BEND.

Refer to Figure 5-1. During the BUST cycle, the virtual address is generated from the BAMX; Memory Management in turn generates the physical address. RACH BUST H is received by the Cache, which starts a CPU cycle if it is idle. During the BUST cycle, the type of transaction is determined by decoding the BSC ROM field (refer to Paragraph 5.1.1).

Cache Address

If the physical address is a Cache reference, SAPN NOT CACHE ADRS is negated and TMCE CONTROL OK is sent to the Cache, which allows the data cycle to start. The clock is stopped in T5 and is restarted upon receipt of the assertion of CCBC MEMSYNC H, by which the Cache indicates completion of its data cycle, i.e., data is ready on read, or taken in on write (refer to Section VI, Cache). At T1, the data from the Cache is strobed into the BR (refer to Chapter 4). In the case of a read-hit (i.e., the word is in the Cache and a Main Memory cycle is not necessary) the clock generally does not stop, because the data is ready and MEMSYNC is asserted before T5.

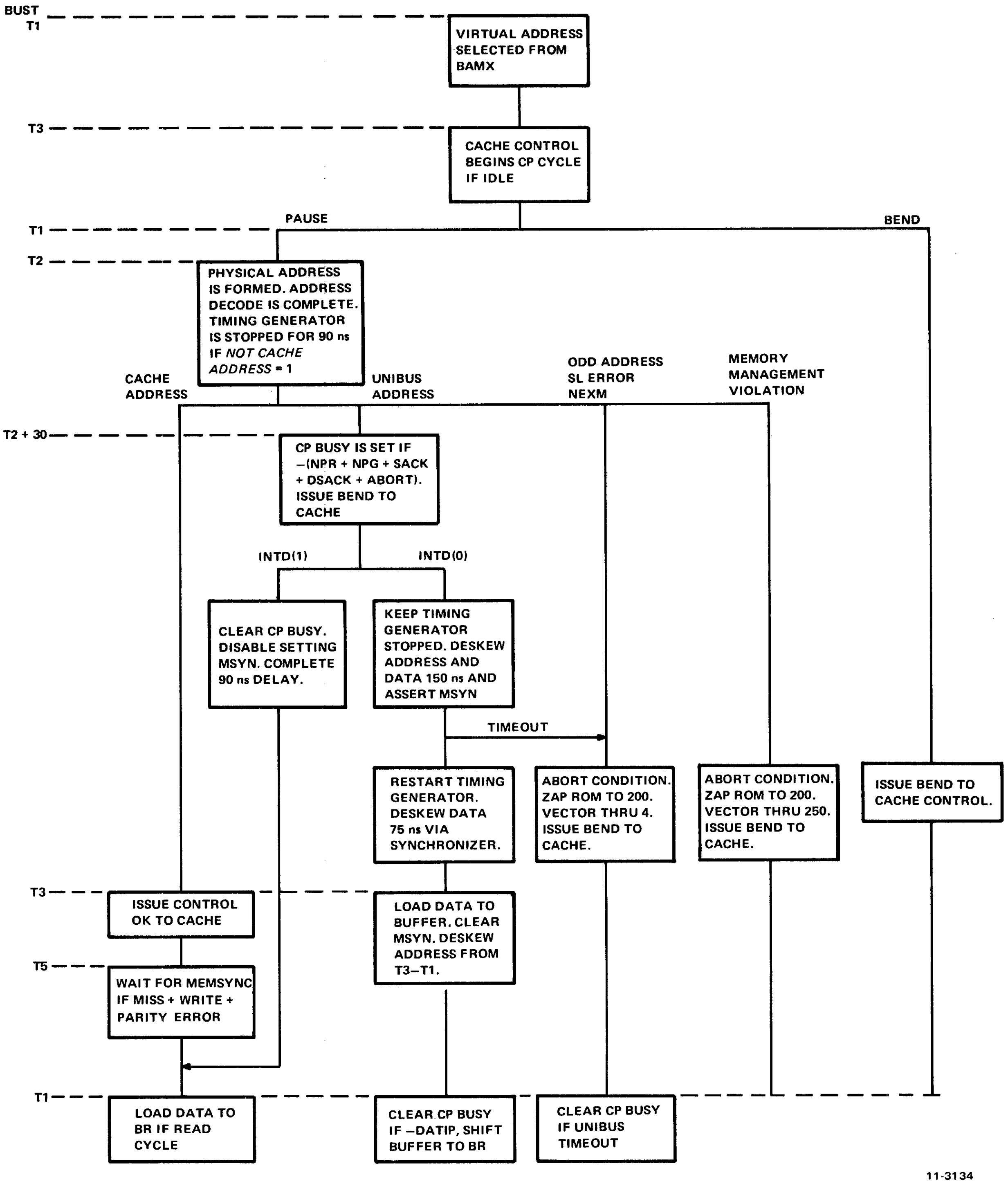

BUST
T1
VIRTUAL ADDRESS SELECTED FROM BAMX
T3
CACHE CONTROL BEGINS CP CYCLE IF IDLE
T1
PAUSE
BEND
T2
PHYSICAL ADDRESS IS FORMED. ADDRESS DECODE IS COMPLETE. TIMING GENERATOR IS STOPPED FOR 90 ns IF NOT CACHE ADDRESS = 1
CACHE ADDRESS
UNIBUS ADDRESS
ODD ADDRESS SL ERROR NEXM
MEMORY MANAGEMENT VIOLATION
T2 + 30
CP BUSY IS SET IF −(NPR + NPG + SACK + DSACK + ABORT). ISSUE BEND TO CACHE
INTD(1)
INTD(0)
CLEAR CP BUSY. DISABLE SETTING MSYN. COMPLETE 90 ns DELAY.
KEEP TIMING GENERATOR STOPPED. DESKEW ADDRESS AND DATA 150 ns AND ASSERT MSYN
TIMEOUT
RESTART TIMING GENERATOR. DESKEW DATA 75 ns VIA SYNCHRONIZER.
ABORT CONDITION. ZAP ROM TO 200. VECTOR THRU 4. ISSUE BEND TO CACHE.
ABORT CONDITION. ZAP ROM TO 200. VECTOR THRU 250. ISSUE BEND TO CACHE.
ISSUE BEND TO CACHE CONTROL.
T3
ISSUE CONTROL OK TO CACHE
LOAD DATA TO BUFFER. CLEAR MSYN. DESKEW ADDRESS FROM T3–T1.
T5
WAIT FOR MEMSYNC IF MISS + WRITE + PARITY ERROR
T1
LOAD DATA TO BR IF READ CYCLE
CLEAR CP BUSY IF −DATIP, SHIFT BUFFER TO BR
CLEAR CP BUSY IF UNIBUS TIMEOUT
11-3134

Figure 5-1 Processor Data Transfers

Unibus Address
If the physical address is a Unibus reference, SAPN UNIBUS ADRS L and SAPN NOT CACHE ADRS H are both asserted and the clock is stopped for a minimum of 90 ns in T2. TMCE CACHE BEND H is asserted and causes the Cache to stop its CPU data cycle. Refer to Section VI.

Thirty ns after the assertion of T2, UBCA CPBSY is set, if all NPRs have been serviced and if no abort is pending.

SCCD INTD REG (1) L is asserted if the Unibus address is a reference to one of the registers that are read on the Internal Data Bus (refer to Chapter 2, Paragraph 2.2.2). If this is the case, CPBSY is reset, MSYN is disabled, the 90 ns S0-S1 delay is completed, the clock is restarted, and the contents of the register that is being referenced is clocked into the BR at the end of the PAUSE ROM state.

If the Unibus reference is not to an INTD register, a Unibus data cycle is executed. The TIG clock is stopped and stays stopped past the 90 ns S0-S1 delay. Address, type of transaction (C1, C0) and, if required, data are put onto their respective Unibus lines and deskewed. MSYN is asserted. The Unibus device that is being addressed executes the transaction and responds by asserting SSYN. The clock is restarted 75 ns after receipt of this signal.

At T3, MSYN is negated and the data is clocked into the PDRJ buffer. At T1 of the next cycle, except in the case of a DATIP, the Unibus lines are cleared by negating CPBSY. If the transaction was a DATIP, CPBSY is not negated, the address lines are not changed (except if the data-out is to be a DATOB, in which case, A00 is changed from 0 to 1 for an odd byte address), the C lines are adjusted, and the data is put on the D lines.

In the case of a DATI or DATIP, the data is clocked into the BR at T1.

Aborts
If an abort occurs, the microprogram forces the ROM address to 200 (ZAP.00). This occurs at T2 of PAUSE for all aborts except parity aborts, which ZAP at T2 of the cycle following the PAUSE. A Memory Management abort vectors through address 250. A parity error abort vectors through address 114. All other aborts vector through address 4. (Refer to Chapter 6.)

5.1.1 Types of Data Transfers

Four types of data transfers are used by the KB11-C. These types are defined by the condition of the Control bits C1 and C0 (TMCE C1 H and TMCE C0 H):

> C1=0, C0=0 – Data-in or DATI. One word of data is transferred to the processor from memory or from the Unibus.
>
> C1=0, C0=1 – Data-in, PAUSE or DATIP. Same as DATI, but a data-out must be executed to the same address immediately following the DATIP. This type of data transfer may be considered as the first part of a read/modify/write operation.
>
> C1=1, C0=0 – Data-out or DATO. One word of data is transferred from the processor to memory or to the Unibus.
>
> C1=1, C0=1 – Data-out, byte or DATOB. One byte of data is transferred from the processor to memory or to the Unibus. The high order byte address is odd and its data is stored in bits 15:08 of a word; the low order byte address is even and its data is stored in bits 07:00 of a word.

The C1 and C0 signals are obtained by decoding the BSC bits as shown on drawing TMCE.

1. When RACC UBSC02 H is negated (low or BSC = 0 – 3) the 74S153 multiplexer is disabled, and both of its outputs are low. Thus, TMCE C1 H and C0 H are low and call for a DATI.
2. When RACC UBSC02 H is asserted (high or BSC = 4 – 7), the BUS COND multiplexer is enabled and its output is a function of RACC UBSC01 and UBSC00, as defined by the table on TMCE.

The BUS CONDITION (BSC) bits of the microprogram ROM determine the type of data transfer by its control of the C lines [TMCE C1 (and C0) H]. The significance of the BSC bits is defined below:

BSC=000 - DATI (data-in), a transfer of one word of data from a slave to the processor.

BSC=001 - SRC1 DATI (SouRCe 1 DATI), a DATI used in odd address error detection to distinguish the first bus operation of source calculation. During a byte instruction, this transaction cannot use an odd address if the source mode is 3, 5 or 7. These are deferred addressing modes and this transaction reads a word containing the address of the operand; this word cannot be odd.

BSC=010 - KERNEL DATI; a DATI is executed, and Memory Management selects the KERNEL PAR/PDR set (refer to Section IV, Memory Management) used in the Trap and Interrupt Service routines to obtain vectored PC and PS from Kernel PAR 0. KERNEL DATI also affects the processor mode bits [PS(15:12)] as explained in Chapter 3.

BSC=011 - SRC2 DATI (SouRCe 2 DATI), a DATI used in odd error detection to distinguish the second bus operation of a source calculation. During a byte instruction, this transaction may use an odd address.

BSC=100 - FC (Floating Point Processor Conditions). Used during FPP Unibus transaction: TMCE C1 H follows the FPP C1 line (FRMJ FP C1 H) and C0 is always negated, since the FPP does only word operations.

BSC=101 - DATO (data-out), a transfer of one word of data from the processor to a slave.

BSC=110 - BSOP1 (BuS OPeration 1): Instruction-dependent bus transaction, specified in execute ROM cycles, common to several instructions that require different types of bus operations. An O/class instruction calls for a DATO, a P/class instruction for a DATIP, and one that is neither O/ nor P/class for a DATI. No instructions are both O/ and P/class. Instruction classes are defined in Chapter 1 and on Flows 3 and 5.

BSC=111 - BSOP2 (BuS OPeration 2). Instruction-dependent bus transaction. If the instruction is a byte instruction, a DATOB (data-out, byte) is executed; if it is not a byte instruction, a DATO is executed.

5.1.2 Types of BUST Cycles

There are two types of BUST cycles: conditional and unconditional, which are described in Chapter 1 (Paragraph 1.2.5.1).

A BUST cycle is one in which the MiSCellaneous (MSC) bits of the microprogram ROM equal 5 or 7:

MSC=5 - CONDITIONAL BUST. This value occurs only in IRD.00 (Flows 1), which generates the A Fork. RACH BUST H is asserted during this cycle, except when the cycle that follows is also a BUST cycle.

MSC=7 - BUST, unconditional.

5.1.3 Types of Pause Cycles

The BUS DELAY (BSD) bits of the microprogram ROM determine the type of Pause cycle to be executed, if any. The significance of the BSD bits is defined below:

BSD=00 - No Pause.

BSD=01 - Interrupt Pause or INTR PAUSE. The Timing Generator is stopped in T2. A Bus Grant is issued. The Timing Generator is restarted by INTR, NO SACK or Passive Release of the Unibus.

BSD=10, BSD=11 - Bus Pause. Used for Internal Data Bus (INTD), Unibus and Cache transactions:

INTD - The Timing Generator is stopped in T2 for 90 ns.

UNIBUS - The Timing Generator is stopped in T2 and restarted after a minimum 90-ns delay by SSYN, Timeout or TMCC ABORT.

CACHE - The Timing Generator is stopped in T5 and restarted by MEM-SYNC or TMCC ABORT.

5.1.4 BEND Cycle

Refer to drawing TMCE. When the ROM BCT (Bus Control) field equals 7, TMCE ROM BEND L is asserted. This condition is indicated by "BEND" on the Flows.

TMCE CACHE BEND H causes the Cache to stop a data cycle. It is asserted by a ROM BEND, when the physical address does not indicate a memory reference (TMCF CACHE ADRS not asserted), by a Memory Management abort (SSRC KT ABORT FLG), by a fatal stack violation (TMCD SL RED) or by an odd address error (TMCC ODD ADRS ERR).

TMCE KT BEND L, when asserted, prevents the modification of the contents of some Memory Management registers and the setting of the KT ABORT FLAG.

5.2 UNIBUS INTERFACE

The Unibus is the transmission medium that interconnects the various components of the PDP-11/70 system, such as peripheral devices, the KB11-C Processor and the Cache Memory via the Unibus Map. The principal connection between the processor and the Cache, however, is direct and does not use the Unibus. Main Memory can only be accessed through the Cache.

The Data Section of the Unibus is used for data transfers between a master device, which controls the transaction, and a slave device, which responds to the master. A master asserts BBSY (Bus Busy); it determines the type of data transfer and is the only device that may assert MSYN (Master SYNc); a slave executes the transaction requested by the master and asserts SSYN (Slave SYNc). The processor is generally a master during Unibus transactions, but in the special case of interrupts, it acts as a slave device.

Only one data transfer may occur at a time on the Unibus, and the priority arbitration logic decides which device may use the data transfer lines on the Unibus. (Refer to Chapter 6, Paragraph 6.3).

5.3 UNIBUS DATA INTERFACE

The KB11-C uses the Data Section of the Unibus for the following types of data transfer:

1. To transmit or to receive data from Unibus devices such as peripheral controller control registers.
2. To access memory via the Unibus Map and then through the Cache; this path is used mainly for diagnostic purposes.
3. To read (only) its control registers (PS, SL, PIR, PIA and PB).
4. To receive a vector during an interrupt transaction.

The transactions listed in (1) and (2) above are identical, and (3) is very similar. These operations are described in this paragraph. The interrupt transaction is explained as part of the Unibus arbitration interface in Chapter 6, Paragraph 6.4.

5.3.1 Unibus Data Transfer Protocol

In order to execute a data transfer on the Unibus, the processor must obey the Unibus protocol:

1. The processor obtains the use of the Unibus from the Unibus priority arbitration logic (refer to Chapter 6).
2. The processor asserts BBSY, thus becoming bus master.
3. The processor defines the slave device with which it wants to communicate. To do this, the processor puts a Unibus address on the A lines [BUS A(17:00) L on SCCL]. Memory Management generates this address (refer to Section IV of this manual).
4. The processor defines the type of data transfer to be execuuted, which is determined by the C lines (BUS C0 L and BUS C1 L on UBCC). Data transfers may be either from the processor to a slave (data-out: DATO or DATOB) or from a slave to the processor (data-in: DATI or DATIP).

5 If the intended data transfer is a DATO or a DATOB, the processor puts the data word or byte on the Unibus D lines [BUS D(15:00) L on PDRE]. Data selection is described in Chapter 2, Paragraph 2.3.2.

6. When these bits (Unibus A, C and D lines) become valid, they are deskewed for 150 ns to allow for decoding in the slave and for variations in bus driver and receiver characteristics (address deskew).

7. The processor then asserts MSYN:

    a. If it is executing a DATI or a DATIP, when the negation of SSYN from the previous Unibus transaction has been received,

    b. If it is executing a DATO or a DATOB, 150 ns after receipt of the negation of SSYN from the previous transaction.

8. The slave receives the assertion of MSYN and either accepts the data from the D lines (DATO or DATOB), or puts the data requested by the processor on the D lines (DATI or DATIP). The slave then asserts SSYN.

9a. DATI or DATIP – Upon receipt of the assertion of SSYN, the master deskews the data received for a minimum of 75 ns. The master then strobes the data and negates MSYN.

9b. DATO or DATOB – The master may negate MSYN upon receipt of the assertion of SSYN. The KB11-C, however, waits 75 ns before negating MSYN.

10. The master waits a minimum of 75 ns after negating MSYN, then removes the address and control bits from the A and C lines. The master then negates BBSY, except in the case of a DATIP, where this signal must remain asserted during the DATO or DATOB that follows the DATIP.

11. The slave typically negates SSYN upon receipt of the negation of MSYN.

12. If the assertion of SSYN is not received within a specified amount of time (Timeout Delay), the instruction is aborted.

5.3.2 Unibus Data Interface

The Unibus data interface is shown on drawings UBCA, UBCB and UBCC. This interface implements the Unibus data transfer protocol.

The description that follows refers to processor Unibus device references, which include the Memory via the Unibus Map. Processor Control Register references differ in some details from these transactions. These differences are described at the end of this paragraph.

5.3.2.1 Unibus Device References

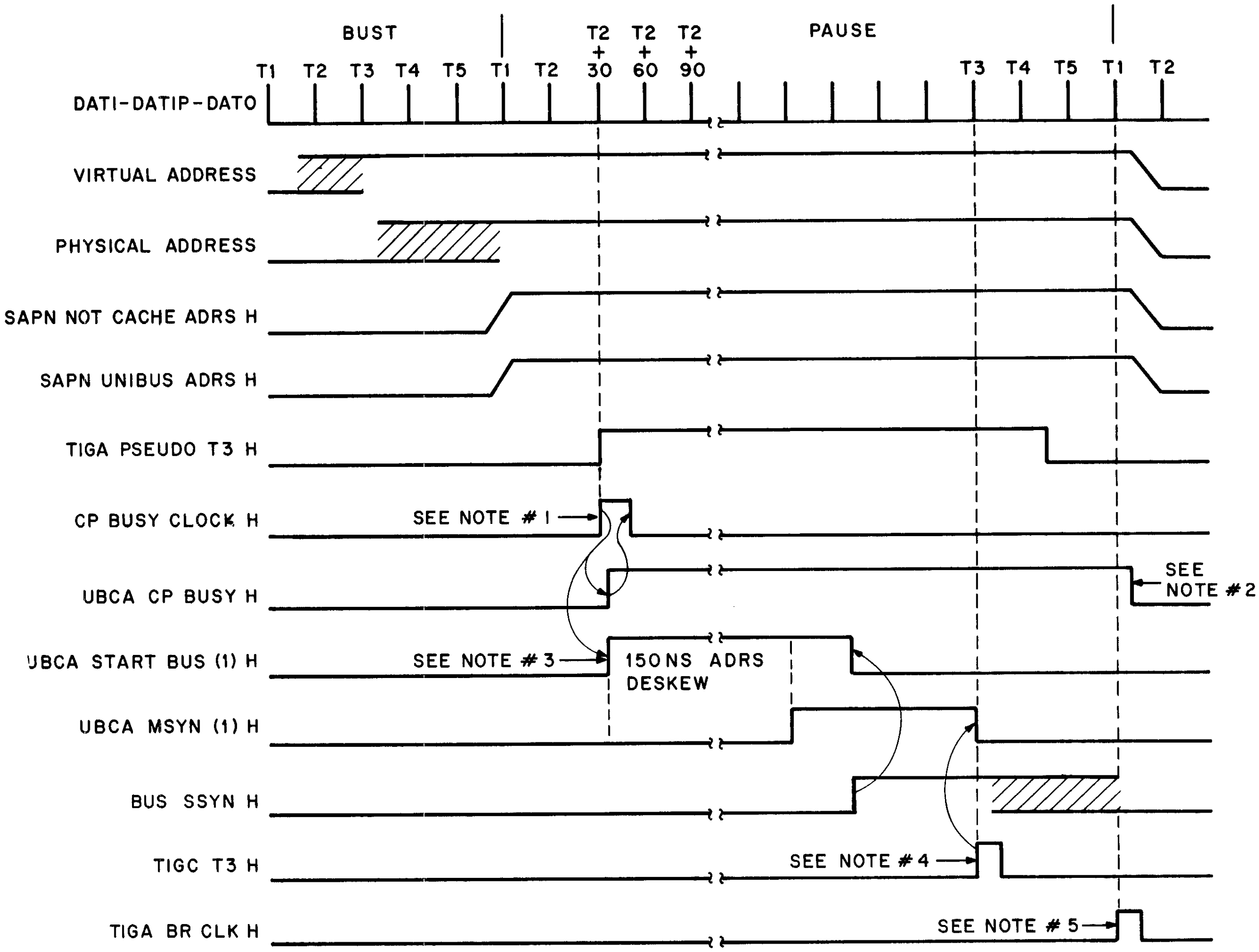
1. During the BUST state, Memory Management generates the Unibus address, which becomes valid by T1 of the PAUSE state. SAPN UNIBUS ADRS L, when asserted, informs the processor that a Unibus transaction is required. The Bus Condition (BSC) ROM bits are asserted during the BUST and during the PAUSE states.

2. During T1 and T2 of the PAUSE state, the Unibus Data Multiplexer (PDRE DMX) selects the input to the Unibus data drivers [BUS D(15:00) L]. Refer to Chapter 2, Paragraph 2.3.2).

3. Refer to drawing UBCA and to Figure 5-2. SAPN UNIBUS ADRS L enables the gate that clocks the UBCA CPBSY flip-flop.

    The TIG clock is stopped in T2 of the PAUSE state (refer to Chapter 4). TIGA PSEUDO T3 H is asserted 30 ns after T2.

    When all NPRs have been serviced, and if no abort is present, and when the previous master has negated BBSY, UBCE CPBSY is clocked and the processor becomes master by asserting BUS BBSY L.



NOTES:

1. Set CP BUSY if -(NPR + NPG + SACK +DSACK + ABORT + BBUSY).

2. CP BUSY is not cleared if DATIP cycle. It is cleared on DATO portion of DATIP/DATO.

3. Used to start DATO address deskew on DATIP/DATO operation.

4. 75 ns data deskew is obtained by 2 stage synchronizer on TIGA. Unibus data is loaded into PDRH buffer register at T3.

5. Address & control are deskewed from T3 to T1. PDRH buffer register loaded to BR at T1.

11-3124

Figure 5-2 Unibus Data Transfers

UBCE CPBSY B H gates the address [BUS A(17:00) on SCCL], the data [BUS D(15:00) on PDRE] and the Control bits (BUS C1 and BUS C0 on UBCC), onto the Unibus.

4. TIGA PSEUDO T3 also clocks and sets UBCE START BUS. If the transaction is a DATO or a DATOB (UBCC C1 B H asserted) and SSYN is negated, the 150 ns address deskew is started. If the transaction is a DATI or a DATIP, the deskew is started without regard to the state of SSYN.

5. Upon completion of the delay, if SSYN is negated, BUS MSYN is asserted by UBCA MSYN.

6. Upon receipt of the assertion of (UBCB) BUS SSYN from the slave, and since MSYN is being asserted by the processor [UBCE MSYN (1) H], UBCB CP SSYN is asserted. This signal clears UBCA START BUS and thus disables the direct-set input to UBCA MSYN (1) H.

7. UBCB CP SSYN L also causes UBCB TIG RESTART to be asserted. This signal causes the clock to be restarted. T3 is asserted 75 ns (minimum) after TIG RESTART is asserted (refer to Chapter 4). T3 clocks the UBCA MSYN flip-flop off and negates BUS MSYN.

   If the transaction is a DATI or a DATIP, the data is clocked into the Bus Buffer Register [PDRJ D(15:00) H] at T3. The 75-ns delay between the assertion of TIG RESTART and that of T3 is the required data deskew.

8. At T1 of the microprogram state that follows the Pause cycle, in the case of a DATI or of DATIP, the data from PDRJ D(5:00) H is clocked into the BR. This is shown as "T6 BR←BUS" of PAUSE on the Flows.

   At the same time (T1) CPBSY is direct-cleared and BUS BBSY L is negated, except in the case of a DATIP, when BBSY must remain asserted until the end of the DATO or DATOB that follows the DATIP. This is controlled by the 74S74 flip-flop on UBCA whose D input is UBCC DATIP L; UBCE MSYN (1) H clocks this flip-flop, which controls the direct-clear input to UBCE CPBSY.

   When UBCA CPBSY B H is negated, the address, data and control bits are removed from the Unibus.

5.3.2.2 Unibus Timeout – If SSYN is not received in response to the assertion of MSYN by the processor within 10 μs a Unibus Timeout occurs.

1. Refer to drawing UBCA. The 74193 binary counter is kept cleared by UBCA MSYN (0) H. When the MSYN flip-flop is set, the counter is free to count up. It is clocked by UBCD FREE CLK (0) H (30 ns pulse every 90 ns) refer to Paragraph 6.4.1). On the 16th clock pulse, a carry is generated which sets the UBCA START TIMEOUT L latch. This counter allows single clock cycle maintenance module operations when referencing Cache registers (or the Cache via the Unibus Map). If the Timeout one-shot was started immediately upon the assertion of MSYN, the Cache, which uses the processor time pulses, could not complete the transaction and Timeout would always occur.

2. Refer to drawing UBCB. The latch starts the timeout 74123 one-shot (10 μs).

   If the assertion of BUS SSYN L is received before the end of the 10 μs, the one-shot is cleared.

If the assertion of BUS SSYN L is not received by the end of the 10 μs the one-shot times out, UBCB TIMEOUT is set and disables the direct-set gate to UBCA MSYN (1) H. TMCC BUS ERROR L and TMCC ABORT H are asserted.

3. Since the clock is stopped in T2 of the Pause cycle (RACB UBSD01 H asserted), TMCC ABORT H asserts UBCB ABORT RESTART H. This signal restarts the TIG clock as in (7) above. The microprogram goes to ZAP.00, thus ending the data transfer cycle.

   UBCB TIMEOUT B H sets TMCD UBUS TIMEOUT H (bit 04 of the CPU Error Register) when TMCC ABORT CLK L is asserted at T3 of PAUSE. The CPU Error Register may be read from address 17 777 766.

5.3.2.3 Control Register Reference - The processor Control Registers (PS, SL, PIR, PIA and PB) are described in Chapter 3. They present a special case of data transfers:

1. They are written directly from the BR, whether they are referenced by Unibus address or by the microprogram. A Unibus cycle is performed as described below when the reference is by Unibus address.
2. The PS can be read either via the Internal Bus (Chapter 2, Paragraph 2.2.2) or via the Unibus. The SL, PIR, PIA and PB can only be read via the Unibus, and not via the Internal Bus.

When referenced by its Unibus address, the register to be read is selected by the DMX. Refer to Chapter 2, Paragraph 2.3.2.

The logic sequence is the same as that for Unibus device references, with the exception that the processor itself must generate SSYN.

Refer to drawing UBCC. SCCE INTERNAL ADRS H is asserted when any one of the addresses in the range of 17 777 770 - 17 777 776 is decoded by Memory Management. These addresses are those of the Control Registers.

Fifty nanoseconds after UBCA MSYN (1) is asserted, BUS SSYN L (UBCC) is asserted. This signal is received by the bus receiver on UBCB and asserts UBCB TIG RESTART H, which restarts the TIG clock.

CHAPTER 6
ABORTS, TRAPS AND INTERRUPTS

An Abort is the non-completion or interruption of a data cycle due to error. This may be a non-recoverable error or, if Memory Management is enabled, a prohibited transaction. Aborts are serviced immediately, prior to the completion of the instruction during which they occur.

A Trap is an interruption of the normal program flow by internal machine conditions. These conditions can be, but are not necessarily errors. A Trap is executed after the instruction during which it occurs is completed.

An Interrupt is similar to a Trap, but is caused by conditions external to the machine. These conditions may be program action (PIR) or external device service requests (BR). Interrupts are controlled by bits 7 - 5 of the Processor Status Word (PSW).

All of the above use the microprogram Service Flows, which are described in Paragraph 6.1. Aborts are explained in Paragraph 6.2, traps and processor interrupts in Paragraph 6.3, and external (Unibus) interrupts in Paragraph 6.4.

6.1 SERVICE FLOWS AND VECTORS

The microprogram Service Flows (Flows 12 and 13) are used during all aborts, traps and interrupts. During these cycles, the PC and PS of the subroutine that is required by the abort, trap, or interrupt are read from memory and the PC and PS of the instruction that caused the entry into the Service Flows are pushed onto the new stack, as determined by the processor mode bits of the new PSW [PS(15:14)].

6.1.1 Vectors

During all aborts, traps and interrupts a Vector is obtained. The vector is the address of the location where the PC for the required subroutine is stored. The vector+2 is the address of the location that contains the new PSW.

During an external interrupt, the vector is provided by the device causing the interrupt, and is read from the Unibus. Refer to Paragraph 6.4. During a power-up, it is read from the Start Vector (SV). During all aborts, internal traps and processor PIR interrupts, it is read from the Trap Vector (TV) logic.

Refer to drawing DAPE. The SV (power-up) is generated by jumpers and is input to the ALU by the BMX. The jumpers may be cut to provide a SV between 00 000 000 and 00 000 174 or between 17 173 200 and 17 173 374.

The TV bits [DAPE TV(01:04) H, TV06 H and TV05*07 H] are controlled by functions generated on TMCB and IRCD. The vectors generated for each function are listed on DAPE. If none of these is asserted, the vector is 4 (TV02).

IRCD decodes the operation code of the IOT, BPT (OPCODE3), EMT and TRAP instructions, which do nothing but generate an interrupt. They are shown on Flows 3, on the A Fork.

6.1.2 CPU Error Register

The CPU Error Register allows the program to determine which abort or trap to location 4 caused entry into the Service Flows. It contains the following bits:

Bit	Name	Function
7	Illegal Halt (trap)	Set when trying to execute a HALT instruction when the CPU is in User or Supervisor mode (not Kernel).
6	Odd Address Error (abort)	Set when a program attempts to do a word reference to an odd address.
5	Non-existent Memory (abort)	Set when the CPU attempts to read a word from a memory location higher than system size register. This does not include Unibus addresses.
4	Unibus Timeout (abort)	Set when there is no response on the Unibus within approximately 10 microseconds.
3	Yellow Zone Stack Limit (trap)	Set when a yellow zone trap occurs.
2	Red Zone Stack Limit (abort)	Set when a red zone abort occurs.

The CPU Error Register is read on the internal data bus (INTD) at address 17 777 766.

6.1.3 Service Flows

6.1.3.1 Entry into the Service Flows – Aborts and Power-up enter the Service Flows through ZAP.00; traps and interrupts enter through BRK.90. The EMT, TRAP and reserved operation codes (from the A Fork, Flows 3) enter through RSD.00. The BPT (OP3) and IOT (also from the A Fork), and the illegal HALT, enter through TR.00. RSD.00 and RSD.10 generate a trap vector (TV) of 4 and shift it left to obtain the correct vector, which is 10. TRP.00 generates the correct TV. These cycles all enter SVC.00 through TRP.10.

6.1.3.2 BRK.90 and ZAP.00 – These two cycles do a BEND, which ends any bus operation that may have been started during the previous cycle.

In addition, ZAP.00 does a BRQ STROBE, which allows setting the CONF after BRK.00 if the HALT switch is down and the S BUS CYCLES INST switch is in S INST.

The BEN06 branch after ZAP.00 checks SSRA PS RESTORE (1) H (Memory Management abort during SVC.70 or SVC.90). Refer to Paragraph 6.2.1.3.

6.1.3.3 BRK.00 and BRK.10 – The INTR PAUSE, during which the vector is read from the Unibus during an external interrupt, occurs during BRK.00. INTR PAUSE is described in Paragraph 6.4. The PC of the instruction preceding the service sequence is stored in the SR.

During BRK.10, the INTR vector is moved into the DR.

6.1.3.4 Branch Enable 13 – The logic that controls Branch Enable 13 (BEN13) is shown on TMCB. All the errors and requests that might be honored to cause an internal trap are ORed to provide an output called TF (and its complement, -TF). The 74H50 gates provide the following two outputs: TMCB PF (0)*(SF+TF) H and TMCB PF (0)*(SF+-TF) H. These outputs control which of four microbranch paths will be followed:

1. PUPF (0) L – If the Power-up flag is set, neither output will be asserted. Microstate PUP.00 (100) will be entered.

2. TF – When the Power-up and Stack Error flags are both cleared [PUPF (0) L and -SERF (1) L] and a trap condition exists, only the TMCB PF (0)*(SF+TF) H output will be asserted. This output causes microstate BRK.80 (140) to be entered.

3. -TF - When the Power-up and Stack Error flags are both cleared and no internal trap conditions are present (-TF), only the TMCB PF (0)*(SF+-TF) H output will be asserted. This causes microstate BRK.20 (120) to be entered.

4. SF - If the Stack Error flag is set and the Power-up flag is not, SERF (1) L will assert both outputs. This will cause the SER.00 microstate (160) to be entered.

The Power-up sequence is described in Paragraph 6.5 and the INTR in Paragraph 6.4.

6.1.3.5 Red Stack Error (SER.00 and SER.10) - The PC and PS pushes in SVC.60 - SVC.80 must be made to locations 0 and 2 of the stack. For this reason, SER.00 and SER.10 set the stack pointer, GR(6), to 4.

After this cycle, the Red Stack Error flows rejoin the flows for all other internal traps by entering BRK.80.

6.1.3.6 BRK.80 and BRK.20 - During BRK.80 the trap vector is read into the DR. The PS is loaded into the BR in both cycles.

The ACKN in BRK.20 clears the INTR flag.

6.1.3.7 BK.30 - This cycle is followed by SVC.00 - SVC.90, which are common to all aborts, traps and interrupts. The ACKN in this cycle sets and clears several functions related to the service flows.

6.1.3.8 Entry into SVC.00 - SVC.00 is entered from either TRP.10 or from BRK.30. At this time, the vector (address of the new PC, which is read first) is in the DR, the old PC is in PCB and in the SR, and the old PS is in the PSW and in the BR.

6.1.3.9 SVC.00 - SV.90 - During these cycles, the PC and PS for the software service routine are read from the Kernel stack during SVC.00 - SVC.20. KERNEL DATI forces Kernel mode but does not change the status bits in the PSW [PS(15:14)]. Refer to Chapter 3.

The new PS is loaded into the PSW during SVC.30. SVC.40 loads the SP into the DR and SVC.50 decrements the SP.

SVC.60 - SVC.90 push the old PS and PC onto the current mode stack as determined by the new PS. If a Memory Management abort occurs during these cycles, the PS RESTORE branch is taken after ZAP.00. Refer to Paragraph 6.2.1.3.

SVC.90 does a BRQ STROBE. It is followed by FET.00.

Table 6-1 shows in detail the movement of data in the processor registers during these cycles.

6.2 ABORTS

Aborts are grouped under three headings in this paragraph: Address, Stack and Parity. The several errors, and their timing, are described under these headings in this paragraph.

6.2.1 Address Errors

An address error causes the Address Error flag (TMCC AERF (1) H) to be set. An address error may be one of the following:

1. Odd Address error,
2. Non-Existent Memory error,
3. Memory Management abort,
4. (Unibus) Timeout error,

provided the bus cycle during which the error occurs is not a push to the Kernel stack.

6.2.1.1 Odd Address Error - An odd address is permissible only during a byte instruction, and then only when the transaction is a SRC1 DATI and the source mode is not 3, 5 or 7, a SRC2 DATI, a BSOP1 or a BSOP2. TMCC ODD ADRS ERR L is asserted when the address is odd (BAMX00=1) and these conditions are not met. The bus cycle is aborted and a trap to 4 is executed.

SRC1 DATI is the first bus operation of source calculation: if the source mode is 3, 5 or 7 (all deferred modes), this transaction reads the address of the operand, which cannot be odd. SRC2 DATI reads the operand, whose address during a byte instruction may be odd.

BSOP1 generates DATIP for a P/class instruction, a DATI for an instruction that is neither P/class nor O/class and a DATO for O/class instructions; no byte instructions are O/class.

BSOP2 generates a DATOB for byte instructions and a DATO for all others.

Table 6-1
Service Flows

μcycle	Type of μcycle	DR	SR	BR	PCA	PCB	BAMX	PS	GR[6]	Comments
Initial Conditions		Vector	old PC	old PS		old PC		old PS		
BRK.30 TRP.10	BUST				old PS		Vector			ACKN new PC to BR from Kernel space.
SVC.00	PAUSE									
SVC.10	BUST	VEC+2*		new PC*			VEC+2			New PS to BR from Kernel space.
SVC.20	PAUSE		old PC*		new PC	old PS*				
SVC.30		old PS*		new PS*	old PS	new PC*		new PS		New PS to PSW
SVC.40				old PS*						New SP (GD[6]) to DR
SVC.50		new SP*							new SP-2	Decrement SP
SVC.60	BUST	new SP-2					new SP-2		new SP-4	First Push: old PS to new Stack; decrement SP
SVC.70	PAUSE									
SVC.80	BUST	new SP-4		old PC*			new SP-4			Second Push: old PC to new Stack
SVC.90	PAUSE									BRQ STROBE if not SERF or PWRF
FET.00										CLEAR FLAGS (SERF or BLOCK STROBE)
FET.10 or BRK.90										

*Occurs at T1, shown as T6 on Flows.

TMCD ODD ADRS ERR L is asserted under the following conditions:

1. The address is odd (BAMX00=1) and the instruction is not a byte instruction (IRCD BY IN H negated). The third gate from the top is asserted in this case.

2. If the address is odd and this gate is not asserted, the instruction is a byte instruction, and either the top or the bottom gate can cause ODD ADRS ERR to be asserted:

   a. If the BSC field calls for a DATI, a KERNEL DATI, a Floating Point Bus Operation or a DATO (BSC=0, 2, 4, or 5), the top gate is asserted;

   b. If the BS field calls for a SRC1 DATI or a DATI (BSC=0 or 1) and a source mode of 3, 5 or 7, the bottom gate is asserted. Note that a DATI causes the top gate to be asserted without regard to the source mode.

6.2.1.2 Non-Existent Memory Error - TMCC NEXM L is asserted when an address is neither a Unibus nor a Cache address. This is determined by ANDing SAPN NOT CACHE ADRS H and SAPN UNIBUS ADRS L. Refer to Section IV of this manual for a description of these functions.

The bus cycle is aborted when reference is made to an address larger than that specified by the System Size Register. The Trap vector is 4 for an NEXM error.

6.2.1.3 Memory Management Aborts - Memory Management aborts are described in Section IV of this manual. SSRC KT ABORT FLG L informs the TMCC logic of such an abort condition. This signal is inhibited when a Stack Limit Red, Odd Address or Non-Existent Memory error is asserted (TMCE KT BEND L). In other words, a Memory Management abort is allowed if no Stack or Address abort is flagged.

KT ABORT asserts TMCC ABORT H and, at T3 of the Pause cycle, sets TMCC SEG ABORTED (1) H, except in the case of a Console operation (UBCF CNSL ACT (0) H).

The SEG ABORTED flip-flop generates the Trap vector for a Memory Management abort. This TV is 250 unless the bus cycle during which the error occurs is a push to the Kernel stack; in this case, the vector is 4 (Stack error, see Paragraph 6.2.2). Refer to TMCB: TMCB SEGT L, when asserted, generates vector 250 on DAPE. SEGT is asserted for an abort when TMCC SEG ABORTED and AERF are both asserted. AERF, however, cannot be asserted when the error is a Stack error (i.e., when TMCC KERNEL R6 is asserted). In this last case, the vector is 4 instead of 25.

PS RESTORE
The Service Flows first fetch the new PC and PS from the vector address; the Kernel stack is used for this operation (SVC.00 - SVC.50). The old PS and PC are then pushed onto the new stack (SVC.60 - SVC.90).

If the new stack is not the Kernel stack, and if Memory Management is enabled and causes an abort during the pushes in SVC.70 or SVC.90, this abort may be a length error, which in this case is a non-Kernel stack error. (A Red Stack error would have occured if the Kernel stack was being used).

1. The microprogram goes to ZAP.00. SSRA PS RESTORE (1) H has been asserted during the push cycle that causes the abort and the microprogram branches to ZAP.10. At this time, the PC and PS of the instruction that caused entry into the Service Flows are in the SR and PCA. PCB and PSW contain the values for the abort, trap or interrupt that was being serviced.

2. ZAP.10 - ZAP.30 restore the PC and PS of the instruction that caused entry into the service routine. The Service Flows are now reentered via BRK.00, BRK.10, and BRK.80. This last cycle fetches the trap vector, which is 250 (Memory Management).

3. BRK.30 - SVC.30 get the Memory Management subroutine PC and PS. This subroutine is typically a Kernel subroutine, and the pushes in SVC.70 and SVC.90 are then to the Kernel stack, and no error should occur.

4. At the end of the Service Flows, control is transferred to the Memory Management software subroutine. This subroutine typically finds the error that caused the abort and corrects the error. In this case, it may allocate more space for the stack.

5. When the software subroutine returns control to the main program, the instruction that originally caused entry into the Service Flows is executed again and causes a new entry into the Service Flows. Since more stack space has been allocated by the software subroutine, the pushes are not successfully executed.

Refer to Section IV of this manual for a description of Memory Management aborts.

6.2.1.4 Timeout Error - UBCB TIMEOUT B L is asserted when a processor Unibus cycle cannot be completed because no device responds to MSYN within approximately 10 μs. The bus cycle is aborted in this case. Refer to Chapter 5, Paragraph 5.3.2. The Trap vector is 4 for a Unibus Timeout error.

A Main Memory timeout on a processor (not a Unibus) cycle is flagged by CCBD CP TIMEOUT L. This signal direct-sets PDRH CACHE PERF L, the Cache parity abort flag, and a Main Memory timeout is processed as a fatal parity error. Refer to Paragraph 6.2.3, Parity Errors. The Trap vector is 114 for a Main Memory timeout error.

6.2.1.5 Timing of Address Error Aborts - Refer to Figure 6-1. The timing diagram shows the approximate time at which TMCC ABORT H is asserted and negated by the several errors. It should be noted that NEXM is derived from the BAMX and is not gated: the times shown in this case indicate the time during which NEXM is valid, i.e., during a Pause cycle.

TMCC ABORT asserts RACA ZAP L at TS2 of the Pause cycle (UBSD01).

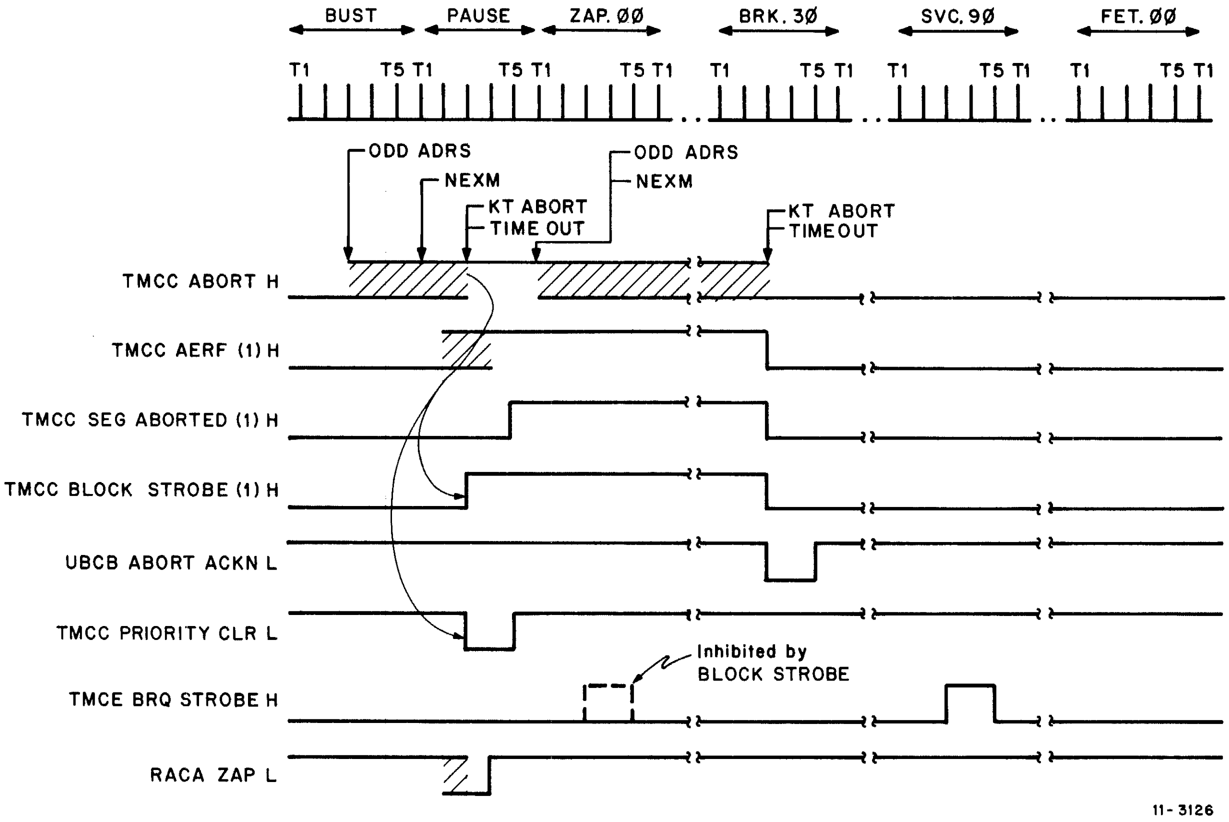


Figure 6-1 Address Error Aborts

TMCC AERF (1) H is set during TS2 of a Pause cycle by any of the address error conditions, if the reference is not to the Kernel stack (KERNEL R6 is negated) and if the bus cycle is not generated by Console action (UBCF CNSL ACT (0) H).

TMCC ABORT direct-sets BLOCK STROBE and asserts PRIORITY CLR during TS3 of the Pause cycle. BLOCK STROBE, while asserted, inhibits BRQ STROBE by asserting TMCC STROBE INH. BLOCK STROBE and PRIORITY CLR prevent any requests previously strobed in from generating vectors during an INTR PAUSE. In this case, since BLOCK STROBE is cleared by its ACKN clock input during BRK.30, the BRQ STROBE during ZAP.00 is inhibited. TMCC PRIORITY CLR clears the request register on TMCA. This allows new requests to be clocked in SVC.90, and a new bran 1 to BRK.90 after FET.00.

AERF and BLOCK STROBE are cleared by ACKN in BRK.30.

6.2.2 Stack Errors

A Stack is an area of memory set aside for temporary storage. Data is added to a stack ("pushed" onto the stack) in sequential order and is retrieved from the stack ("popped" from the stack) in reverse order. A stack starts at its highest address and expands toward its lowest address as data is added to it.

The address of the last valid item pushed onto the stack is stored in a general register which is called the Stack Pointer (SP). When an item is pushed onto a stack, the SP is first decremented to the next lower address, then the item is written using the SP as the address. When an item is popped from a stack, the item is read using the SP as the address, then the SP is incremented to the next higher address. Further details on stacks and their use are included in Chapter 9 of the *PDP-11/70 Processor Handbook*.

There are three Hardware Stacks, one each for Kernel, Supervisor and User modes. The particular register (R6) for each mode is the SP for that mode's hardware stack. These stacks are word-oriented and the SPs can only be incremented or decremented by 2.

The Kernel stack differs from the other two in that it is hardware-protected.

The Supervisor and User stacks are not protected by hardware, but may be checked by Memory Management and appropriate software. Refer to Paragraph 6.2.1.3 (PS Restore).

A stack error is one which occurs during a push to the Kernel stack. When such a push occurs, TMCC KERNEL R6 (1) H is asserted. If an error occurs during this push, TMC SERF (1) H (the Stack Error flag) is set.

A stack error may be any of the address errors listed in Paragraph 6.2.1 or a Stack Limit Red error.

The above errors all cause aborts. Stack Limit Yellow is a stack error, but traps instead of aborting. Refer to Paragraph 6.2.2.2.

Both SL YEL and SL RED vector to 4, with the exception of an SL RED that occurs during a power fail. Refer to Paragraph 6.5.1.

6.2.2.1 Kernel R6 – TMCC KERNEL R6 (1) H is a J-K flip-flop that is clocked at T4. It is set during a data-out (including DATIP) BUST cycle if the reference is to the Kernel stack. It is reset during the Pause cycle that follows the BUST.

The J input to the flip-flop is a 74S64 gate. All the OR inputs to this gate must be asserted if the output of the gate is to be high (asserted).

1. The second gate from the top is asserted during a BUST cycle that calls for any type of data transfer except a DATI.

2. The third gate is asserted when General Destination Register Set 0 is addressed (GRAC GRA3 L is asserted when GD Set 1 is addressed).

3. The top and bottom gates are asserted in two cases:

   a. When BAX = 0 or 2 and the BAMX selects the contents of either the DR or the SR (RACB UBAX00 H is negated) and General Destination Register 6 is selected (GRAC GD6 L is asserted). In this case, GD register 6, Set 0, is used as the address for a data-out operation: this is a push onto the Kernel stack. During these cycles, the General Registers are addressed using the destination field (GD[DF]) on the Flows, and the description of the cycle includes the sentence: "Check Stack Limit."

   b. During a JSR, the contents of the source field register are pushed onto the stack. This is done during JSR.30 (Flows 11) where BCT = 5 (STACK REFerence). If PDRD PS14 (0) H is asserted, the processor is in Kernel mode, and the push is to the Kernel stack. The 74S20 NAND gate is asserted, as are the top and bottom gates of the 74S64.

      The K input to KERNEL R6 (1) H is TMCE PAUSES H, which is asserted during Bus Pauses (BSD = 2 or 3) to clear the flip-flop.

6.2.2.2 Stack Limit Errors – The lower limit of the Kernel stack is set by program control of the Stack Limit Register (SL). Any bus cycle that does a push beyond this lower limit is aborted (Stack Limit RED or SL RED). A warning zone of 16 words exists where any push causes a trap (Stack Limit YELlow or SL YEL).

The default boundary for stack addresses is 400. This is the case when the SL contains 0. The Stack Limit Register allows this lower limit to be raised, providing more address space for interrupt vectors or other data that should not be destroyed by the program. This limit may be varied in increments of 400_8 words, up to a maximum virtual address of 177 400 by modifying the content of the Stack Limit Register (SL). This register contains eight bits and can be addressed as a word at location 17 777 774, or as a byte at location 17 777 775. The register is accessible to the processor and Console, but not to any Unibus device. The 8 bits, PDRC SL(07:00), contain the stack limit information and are compared with BAMX(15:08). These bits are cleared by System Reset, Console Start, or the RESET instruction. The lower 8 bits are not used. Bit 8 corresponds to a value of $(400)_8$ or $(256)_{10}$.

Stack Limit Violations
When instructions cause a stack address to exceed (to go lower than) a limit set by the programmable Stack Limit Register, a Stack Violation occurs. There is a Yellow Zone (grace area) of 16 words below the Stack Limit which provides a warning to the program so that corrective steps can be taken. Operations that cause a Yellow Zone Violation are completed, then a bus error trap is executed. The error trap, which itself uses the stack, executes without causing an additional violation, unless the stack has entered the Red Zone.

A Red Zone Violation is a Fatal Stack error. (Odd stack or non-existent stack are the other Fatal Stack errors). When detected, the operation causing the error is aborted, the SP is set to point to address 4, and a bus error occurs. The old PC and PS are pushed into location 0 and 2, and the new PC and PS are taken from locations 4 and 6.

Stack Limit Addresses
The contents of the SL are compared to the stack address during a push to determine if a violation has occurred.

If the contents of the SL are zero:

Yellow Zone = 340 – 376: execute, then trap;

Red Zone = 000 – 336: abort, then trap.

If the contents of the SL are greater than zero:

Yellow Zone = (SL)+(340 – 376): execute, then trap;

Red Zone =(SL)+(336): abort, then trap.

Stack Limit Yellow
Refer to Figure 6-2. PDRC STACK LIMIT H is asserted when the high order eight bits of the virtual address [BAMX(15:08)] equal the contents of the Stack Limit Register [PDRC SL(07:00)].

When bits 7 – 5 of the virtual address are all ones, the value of bits 7 – 0 of the address is between 377 and 340. TMCD YEL ZONE H is asserted.

Thus, when PDRC STACK LIMIT H and TMCD YEL ZONE are both asserted, a Yellow Zone stack violation exists.

TMCD SL YEL (1) H is then set at T2 + 15 ns of a Pause cycle (UBSD01 H) that is pushing onto the Kernel stack (KERNEL R6).

SL YEL is cleared by setting the SERF flip-flop (ACKN in BRK.30); SERF inhibits the BRQ STROBE in SVC.90.

Stack Limit Red
Refer to Figure 6-2. If a Yellow Zone condition exists and the address is further decremented, TMCD YEL ZONE goes low and the bottom gate of TMCD SL RED is asserted. SL RED is a latch, and is set by this gate at T5 of the BUST cycle.

T5 is gated with KERNEL R6. This gate is disabled if BLOCK STROBE and SERF are both asserted, i.e., SL RED cannot be asserted again during the pushes to 0 and 2 in SVC.60 and SVC.80.

SCCE STACK OVERFLOW H is asserted if the virtual address equals 177 776. This gate asserts SL RED in the case that the SP is decremented from 0 to protect the Processor Status word.

PDRC RED ZONE H is asserted when the virtual address is less than the SL.

SL RED is cleared by ABORT ACKN in BRK.30. Figure 6-2 is a summary of the conditions that cause a Stack Limit error.

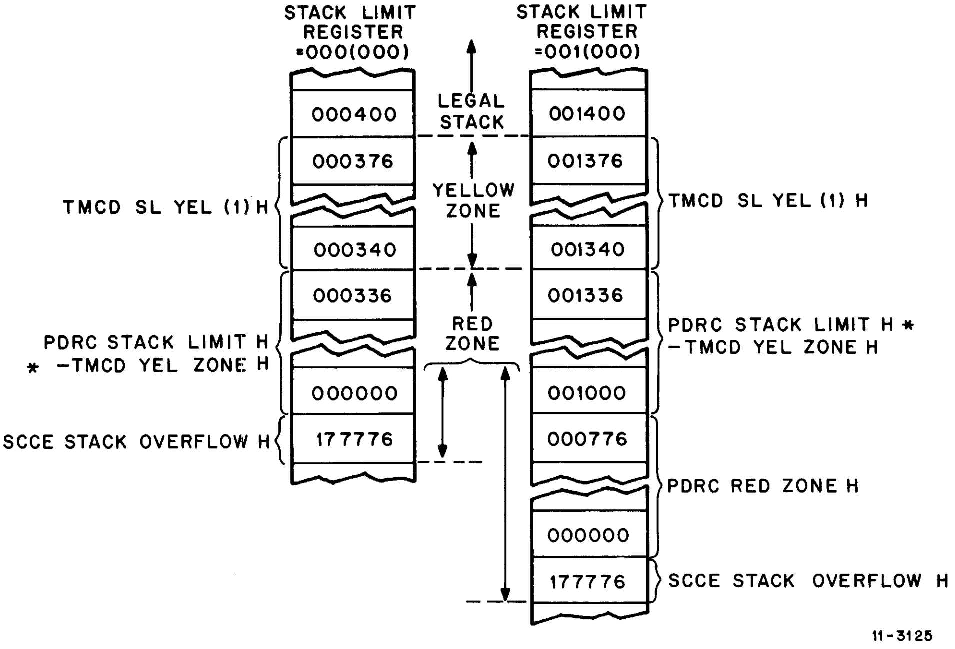


Figure 6-2 Examples of Stack Limit

6.2.2.3 Timing of Stack Error Aborts - Refer to Figure 6-3. The timing for stack errors is similar to that for address errors, with the following exceptions:

1. TMCC KERNEL R6 (1) H is asserted at T4 + 15 ns of BUST and cleared at T4 + 15 ns of PAUSE.
2. KERNEL R6 causes TMCC SERF (1) H to be set (instead of AERF).
3. Since SERF is asserted, BLOCK STROBE and therefore PRIORITY CLR are asserted until TS3 of FET.00, when SERF and BLOCK STROBE are cleared by CLEAR FLAGS (BCT=3, TMCC).

BRQ STROBE is thus inhibited, not only during ZAP.00, but also during SVC.90, thus guaranteeing the execution of the first instruction of the error subroutine before any other error can be processed.

6.2.3 Parity Errors

6.2.3.1 Description - A parity error may be detected either by the Cache or by a Unibus device.

Cache parity errors are either "hard", if bad parity is detected in the word requested by the processor, or "soft", if the Cache can recover without processor intervention. Hard errors are signalled by the assertion of CCBJ PARITY ABORT H and cause the processor to abort; soft errors are signalled by CCBJ PARITY TRAP H and cause a trap.

It should be noted that Main Memory Timeout is included in the Cache parity error logic: CCBD CP TIMEOUT L direct-sets the flip-flop (PDRH CACHE PERF) that stores CCBJ PARITY ABORT.

All Unibus parity errors are hard and cause an abort; a device asserts BS PB L (UBCB) when it senses a parity error (BUS PA is never asserted).

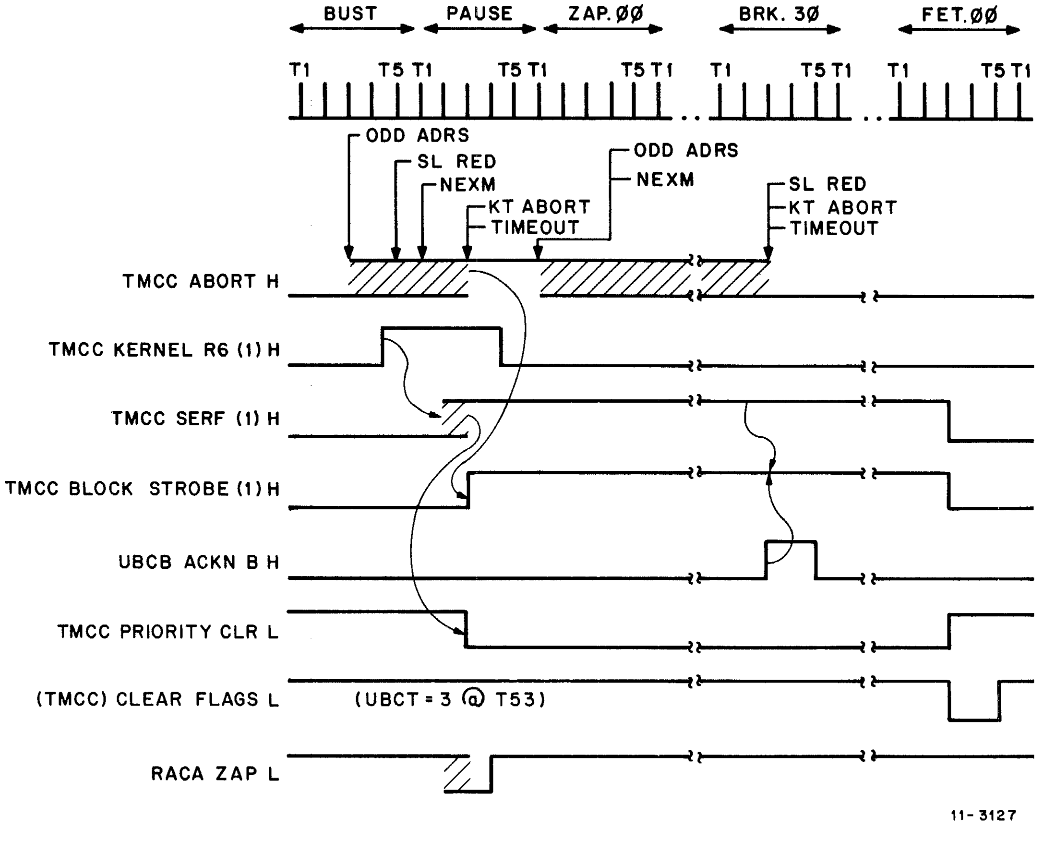


Figure 6-3 Stack Error Aborts

The vector for both parity aborts and parity traps is 114. UBCB PARITY ERR L enables the Trap Vector on DAPE.

6.2.3.2 Timing of Parity Error Aborts – Refer to Figure 6-4.

Unibus Parity Error

1. BUS PA L and BUS PB L are clocked into the 74S175 flip-flop (UBCB) at T3 of a Unibus Pause cycle (MSYN negated). At T1 of the cycle following the PAUSE, the 74S10 NAND gate is enabled by the negation of TMCE PAUSES L. The output of the NAND gate is asserted if PB is asserted and PA negated (parity error), and if TMCE PAUSEs is negated. PAUSES prevents the assertion of the NAND gate and of PE ABORT during the PAUSE state.

   UBCB UBUS PAR ERR H resets trap request flip-flop on CCBK (Refer to Section VI, Chapter 4).

2. This Unibus parity error signal is ORed with the Cache parity error signal and input to UBCB PE ABORT L. This NAND gate is enabled by the 74S74 flip-flop, which is set during all cycles that follow a Pause.

3. PE ABORT then asserts TMCC ABORT H, BLOCK STROBE (1) H and PRIORITY CLR L, as in other aborts. RACA ZAP L is then asserted by PE ABORT at TS2 of the microprogram state following the PAUSE cycle.

4. BLOCK STROBE is cleared by UBCB ACKN during BRK.30, which allows a BRQ STROBE in FET.00.

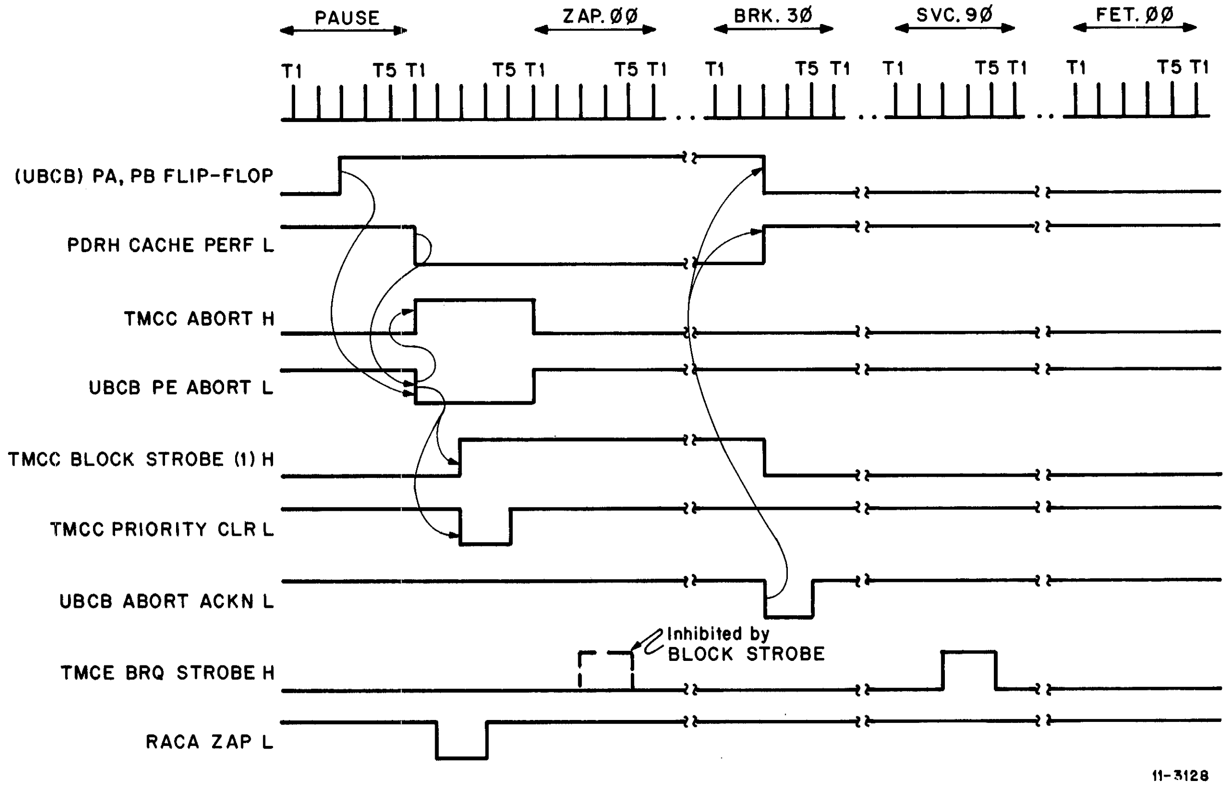


Figure 6-4 Parity Abort

Cache Parity Error

1. CCBJ PARITY ABORT H is clocked into PDRH CACHE PERF L by TIGA CLK BRA H, which occurs at T1 of the cycle following a Pause. (CCBJ CP TIMEOUT direct-sets CACHE PERF, thus combining the Cache Parity and Timeout errors).

2. This Cache parity error signal is ORed with the Unibus parity error signed and input to UBCB PE ABORT L. This NAND gate is enabled by the 74S74 flip-flop, which is set during all cycles that follow a Pause.

3. PE ABORT then asserts TMCC ABORT H, BLOCK STROBE (1) H and PRIORITY CLR L, as in other aborts. RACA ZAP L is then asserted by PE ABORT at TS2 of the microprogram state following the Pause cycle.

4. BLOCK STROBE is cleared by UBCB ACKN during BRK.30, which allows a BRQ STROBE in FET.00.

6.3 TRAPS AND INTERRUPTS

Trap and interrupt requests are clocked into the request storage (or "Q") register on TMCA and TMCB. TMCE BRQ CLK H clocks these requests into the register at least once during the execution of each instruction (with the exception of SPL). BRQ CLK may be inhibited by an abort which may also clear the Q register in order to give highest priority to the abort (refer to Paragraph 6.2).

The requests are examined by the priority arbitration network (TMCA, TMCB), which allows only the highest priority request to be honored. One of the signals in the Output column of Table 6-2 is then asserted.

If the request is not an external interrupt (UBCD EXT BRQ L) the ENB VEC flip-flop is set and enables the gates that generate the vector addresses for the requests. Table 6-3 lists the requests, the gates enabled and the vectors that are generated.

All instructions except SPL end with a BEN12 branch to microaddress 240. If TMCB BRQ TRUE L is asserted, BRK.90 is entered instead of a Fetch cycle, and the Service Flows are executed, followed by the subroutine determined by the new PC.

6.3.1 Illegal Halt

A trap to 4 is executed, instead of a HALT at the end of a HALT instruction, if the processor is not in Kernel mode, as determined by PS14. If the mode is Kernel, the Console Flag is set and a branch to CON.00 is executed.

Refer to drawing TMCE. During HLT.10 (Flows 3), MSC=3, SET CONF if Kernel mode; TMCE SET HALT H is asserted. At TS3, if the processor is in Kernel mode [PDRD PS14 (0) H], the Console Flag is set and the processor halts. This is shown on Flows 3 as CONF←1 IF PS14(0).

If, on the other hand, PS14=1 (Supervisor or User modes), BEN06, which examines BR14, causes a branch to TRP00. (The PS is stored in the BR during HLT.00.)

6.3.2 Console Flag

TMCA CONF (1) H causes a processor HALT by causing it to branch to CON.00 (Flows 14). Refer to Section III (Console) of this manual.

6.3.3 Cache Parity Trap

CCBJ PARITY TRAP H is asserted by the Cache if it detects a non-fatal parity error, i.e., one which does not affect the processor bus cycle in progress. Refer to Section VI of this manual for a complete description.

6.3.4 Memory Management Traps

Refer to Section IV (Memory Management) of this manual.

6.3.5 Yellow Zone Trap (SL YEL)

Refer to Paragraph 6.2.2.

6.3.6 Power Down Trap (PDNF)

Refer to Paragraph 6.5.

6.3.7 FP Exception Trap

Refer to Floating Point Processor Manual.

Table 6-2
Processor Service in Order of Priority

Order	Condition	Input	Output*	Result*
1	console flag	UBCF STOP L	TMCA CONF (1) H	do console control function
2	cache parity	CCBJ PARITY TRAP H	TMCB PART L	trap (114)
3	memory management traps	SSRD MEM MGMT TRAP L	TMCB SEGT L TMCA HONOR SEGT H	trap (250)
4	warning stack violation	TMCD SL YEL	TMCA HONOR SLY H	trap (4)
5	power fail	UBCE PDNF (1) H	TMCA HONOR PWRF L	trap (24)
6	floating-point exception trap	FRHH FP EXC TRAP L	TMCA HONOR FPTRAP L	trap (224)
	CP LEV 7			
7	priority interrupt request PIRQ7	PDRD PIR15 (1) H	TMCA HONOR PIR7 L	trap (240)
8	bus request, level 7 interrupt	BUS BR7 L	TMCA HONOR BR7 L	interrupt
	CP LEV 6			
9	priority interrupt request PIRQ6	PDRD PIR14 (1) H	TMCA HONOR PIR6 L	trap (240)
10	bus request, level 6 interrupt	BUS BR6 L	TMCA HONOR BR6 L	interrupt
	CP LEV 5			
11	priority interrupt request PIRQ5	PDRD PIR13 (1) H	TMCA HONOR PIR5 L	trap (240)
12	bus request, level 5 interrupt	BUS BR5 L	TMCA HONOR BR5 L	interrupt
	CP LEV 4			
13	priority interrupt request PIRQ4	PDRD PIR12 (1) H	TMCA HONOR PIR4 L	trap (240)
14	bus request, level 4 interrupt	BUS BR4 L	TMCB HONOR BR4 L	interrupt
	CP LEV 3			
15	priority interrupt request PIRQ3	PDRD PIR11 (1) H	TMCB HONOR PIR3 L	trap (240)
	CP LEV 2			
16	priority request PIRQ2	PDRD PIR10 (1) H	TMCB HONOR PIR2 L	trap (240)
	CP LEV 1			
17	priority request PIRQ1	PDRD PIR09 (1) H	TMCB HONOR PIR1 L	trap (240)
18	T bit set and not RTT	PDRD PS04 (1) H and -(IRCD RTT L)	TMCB HONOR T L	trap (14)

* Only if no higher priority request has been received.

Table 6-3
Trap Vectors Enabled

Trap Request Honored	Output	Trap Vector*
TMCB PART L	UBCB PARITY ERR L	114
TMCA HONOR FPTRAP H	TMCB FPTRAP L	244
TMCA HONOR SEGT H	TMCB SEGT L	250
TMCA HONOR PWRF H	TMCB PWRF L	24
TMCB HONOR T H	TMCB TOK L	14
TMCB HONOR PIRQ H (OR of PIR ⟨7:1⟩)	TMCB PIRQ L	240

* Trap vector generator is shown on drawing DAPE.

6.3.8 Program Interrupt Request

The Program Interrupt Request (PIRQ) Register allows a program to schedule the execution of various subprograms, according to a priority scheme, at the same time allowing various levels of hardware interrupt priority to interact with the software priority levels. The register stores interrupt requests set by transferring request data to the PIRQ, and provides information about the requests through encoded data transferred from the PIRQ.

A request is booked by setting one of the bits 15 – 9 (for PIR 7 – PIR 1) in the Program Interrupt Register at location 17 777 772. The hardware sets bits 7-5 and 3-1 to the encoded value of the highest PIR bit set. This Program Interrupt Active (PIA) is used to set the Processor Level and also to index through a table of interrupt service routines for the seven software priority levels. Figure 6-5 shows the layout of the PIR.

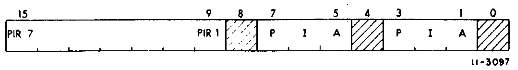


Figure 6-5 Program Interrupt Request Register

When the PIR is granted, the processor traps to location 240 and picks up the PC in 240 and the PSW in 242. It is the interrupt service routine's responsibility to queue requests within a priority level and to clear the PIR bit before the interrupt is dismissed.

Refer to drawing PDRD. PIR(15:09) (1) H is loaded from BR(15:09) when MSYN is set and if the PIR address is decoded (SCCE PIR ADRS H). The clock signal is TMCF CLK PIR H. The PIR bits are encoded by the 9318, which generates PDRD PIA(02:00).

Both PIR and PIA are read on the Internal Data Bus (INTD). The PIR is read as bits 15:09, and the PIA is repeated in bits 07:05 and 03:01. Bits 7 - 5 allow the program to move the PIA into the processor status register and thus set the processor priority to the level of the request honored, if desired. This locks out all requests on the same level or below. Bits 3 - 1 can be used as an index constant in dispatching to an interrupt service routine for the appropriate priority level request.

6.3.9 External Interrupt (BUS BR)

Refer to Paragraph 6.4.

6.3.10 T Bit Trap

When the T bit is set (refer to Chapter 3), and if there are no higher priorities, a trap to 14 occurs through RSD.00.

Detailed information on the execution of the T bit trap is contained in the *PDP-11/70 Processor Handbook*.

6.4 UNIBUS ARBITRATION AND INTERRUPT INTERFACE

An NPR transfer is a data transfer between a Unibus device and memory; the processor is not involved in this transfer except to the extent that it cannot use the Unibus or memory during its execution. AN NPR transfer can be executed at any time that the processor is not using the Unibus.

A BR transfer is a data transfer during which a vector is transmitted to the processor by a Unibus device, which requires the execution of a service routine by the processor. The vector is the address of the PC that is to be used for this subroutine. A BR can only be executed at the end of an instruction.

The priority arbitration network (Paragraph 6.3) examines the requests received from the Unibus, compares their priorities against that of the processor, and decides which device may become master when the Unibus is released by the current master.

The Unibus Request signals received by the KB11-C are listed below:

> *NON-PROCESSOR REQUEST, BUS NPR L (UBCD):* A signal from an asynchronous running device requesting the use of the data section of the bus, sent to arbitrator by a device that requires the use of the Unibus in order to execute data transfers. These transfers are made without active participation by the processor. They do not affect processor operations, except to the extent that Unibus devices using the bus for a data transfer can force the processor to wait in the PAUSE state until all NPRs have been serviced.
>
> NPR transfers are executed between processor Unibus cycles (i.e., when the processor is not using the Unibus), and not necessarily after completion of an instruction. NPRs may be asserted at any time that the device is ready to start a data transfer. NPRs have a higher priority than processor data transfers or than any of the BR lines.
>
> *BUS REQUEST, BUS BR7 L - BUS B4 L (TMCA, TMCB);* A signal from an asynchronous running device, requesting the use of the data section of the bus. Typically, one of these signals is sent to the arbitrator by a device that requires the use of the Unibus to transmit an interrupt vector to the processor.
>
> An interrupt is a transfer of control to a subprogram that handles device or task servicing. An interrupt vector points to the address of this subprogram; the vector is transmitted to the processor during an interrupt (INTR) transaction.
>
> Interrupt transactions require processor action, and can only be executed after the current instruction is completed.
>
> A BR may be asserted at any time that the device is ready to interrupt the processor, but cannot be serviced until the processor is ready to do so. BRs have lower priority than NPRs and than a processor priority of the same level (7 - 4).

Priorities permitting, the **KB11-C** responds to these requests by asserting one of the following GRANT signals:

> *NON-PROCESSOR GRANT, UBCD PROC NPG H* - unless INIT, RESET or ACLO are asserted, or during a read/modify/write (UBCC DATIP L), or if the Console HALT/ENABLE switch is in HALT and the S INST/S BUS CYCLE switch is in S BUS CYCLE. During a DATIP, no grants are issued in order to minimize NPR latency.
>
> *BUS GRANT, UBCD PROC BG7 - BG4 H* - if the priority arbitration network has asserted the corresponding TMCA HONOR BR7 - BR5 L.

Only one grant (NPG or BG) may be asserted at a time.

The requesting device, upon receipt of a grant, asserts BUS SACK L, then negates its request. When the assertion of SACK is sensed (UBCD), the grant is negated. No grants may be asserted while SACK is asserted. When the requesting device negates SACK, a new grant may be issued.

If no device responds to a grant by asserting SACK within 10 μs, UBCD NO SACK (1) H is asserted, forces SACK, thus allowing the assertion of a new grant. A NO SACK timeout does not cause a trap or abort. It should be noted that some Unibus terminators (e.g., 9302), when used at the end of the Unibus that is opposite to the processor, receive NPR (if no device has accepted it), and assert SACK. The Timeout delay is thus not used.

An NPG may only be used by a device for data transfer. No interrupts are allowed on an NPG, and the processor is not affected by an NPR transaction.

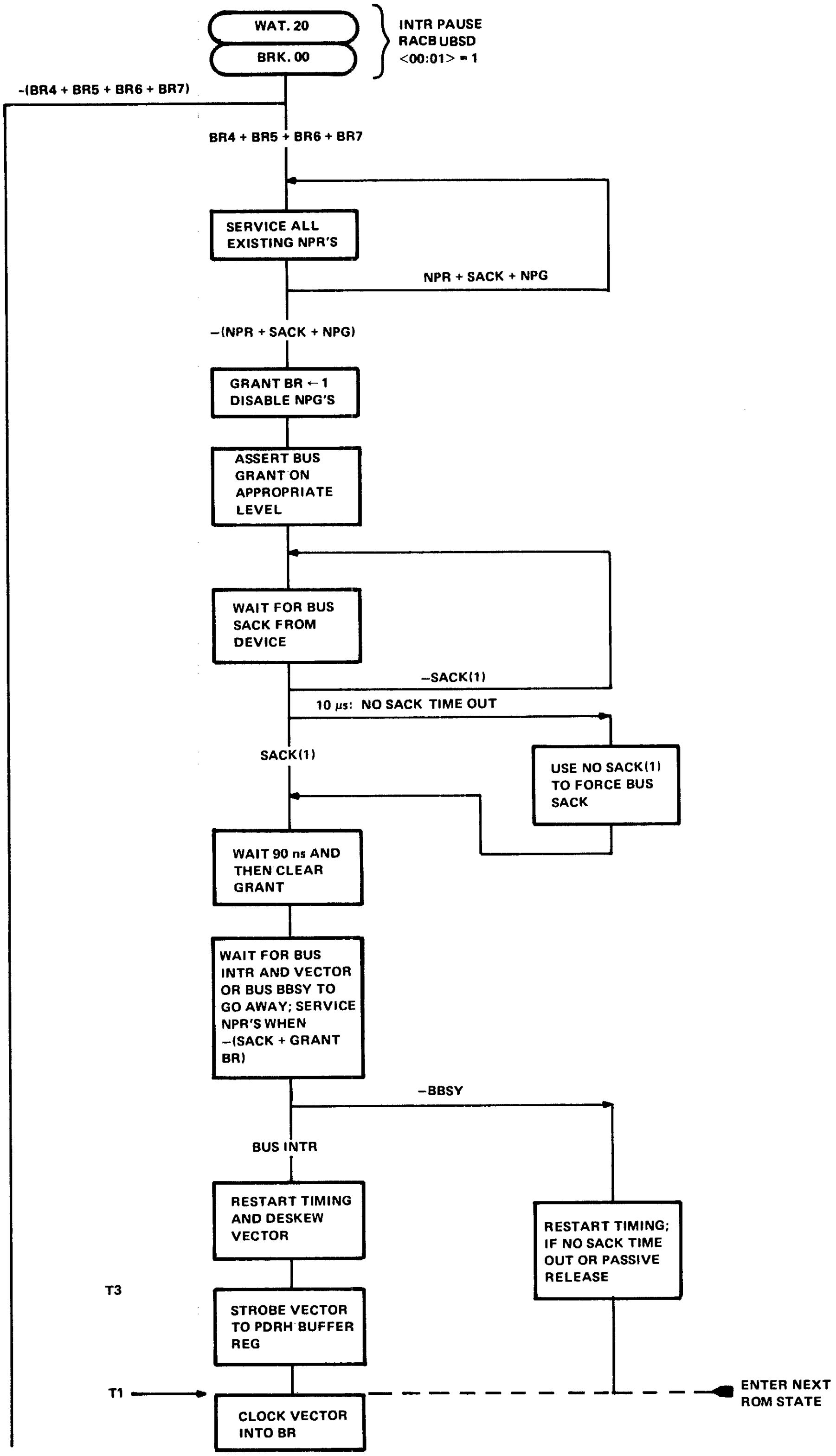

WAT. 20
BRK. 00
INTR PAUSE
RACB UBSD
<00:01> = 1
-(BR4 + BR5 + BR6 + BR7)
BR4 + BR5 + BR6 + BR7
SERVICE ALL EXISTING NPR'S
NPR + SACK + NPG
–(NPR + SACK + NPG)
GRANT BR ← 1
DISABLE NPG'S
ASSERT BUS GRANT ON APPROPRIATE LEVEL
WAIT FOR BUS SACK FROM DEVICE
–SACK(1)
10 μs: NO SACK TIME OUT
SACK(1)
USE NO SACK(1) TO FORCE BUS SACK
WAIT 90 ns AND THEN CLEAR GRANT
WAIT FOR BUS INTR AND VECTOR OR BUS BBSY TO GO AWAY; SERVICE NPR'S WHEN –(SACK + GRANT BR)
–BBSY
BUS INTR
RESTART TIMING AND DESKEW VECTOR
RESTART TIMING; IF NO SACK TIME OUT OR PASSIVE RELEASE
T3
STROBE VECTOR TO PDRH BUFFER REG
T1
CLOCK VECTOR INTO BR
ENTER NEXT ROM STATE
11-3136

Figure 6-6 BR –Interrupt Sequence

A BG, on the other hand, is used for an interrupt. Refer to Figure 6-6. When an interrupt is sensed, the microprogram branches to the BRK sequence (Flows 12). BRK.00 is the INTR PAUSE cycle (BSD=1) in this sequence. A similar cycle, WAT.20, is part of the WAIT instruction. The INTR PAUSE cycle is the only condition in which the processor acts as a Unibus slave (i.e., asserts SSYN).

During the INTR PAUSE cycle, the clock is stopped in T2 if an external interrupt is to be serviced (BR4+BR5+BR6+BR7). After all pending NPRs have been serviced, the Bus Grant (BG) is asserted on the level corresponding to the level of the request that is to be serviced.

When the requesting Unibus device receives the BG it acknowledges this by asserting SACK and then negating its BR. The device that asserts SACK asserts BBSY when the previous master negates it.

The processor negates the BG 90 ns after receiving the assertion of SACK; typically, a device asserts INTR and the vector just before it negates SACK.

Two parallel and generally unrelated sequences now occur:

1. The assertion of INTR is received from the Unibus. The clock is restarted at T3, the vector is strobed, and SSYN is asserted. The Unibus device negates INTR when it receives the assertion of SSYN. The processor negates SSYN when it receives the negation of INTR.

2. The negation of SACK is received and, after a minimum wait of 90 ns, allows NPRs to be processed.

6.4.1 Unibus Arbitration Interface Logic

The Unibus arbitration interface logic is controlled by UBCD FREE CLK which consists of the two 74S112 J-K flip-flops clocked by TIGC TF L. The FREE CLK generates a 30-ns wide pulse within a period of 90 ns. The D flip-flops (74S74S) on UBCD use the inverted output of this clock, while the J-K flip-flops (74S112s) other than those that make up the clock use the non-inverted output. Thus, the two sets of flip-flops are clocked at the same time. Figure 6-7 shows the output of the FREE CLK.

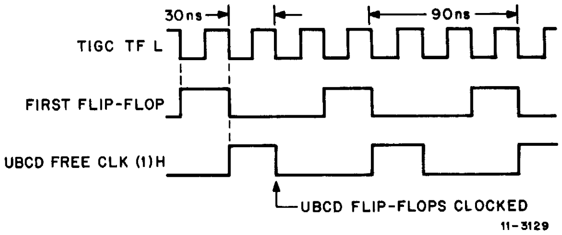


Figure 6-7 UBCD Free Clock

The relationship between the FREE CLK and the TIG timing pulses (T1-T5) and time states (TS1-TS5) is such that the leading or trailing edge of the FREE CLK and the first FREE CLK flip-flop outputs always coincide with the leading edge of T1-T5 and TS1-TS5. There is no other relationship to the TIG clock.

6.4.2 NPR-NPG Sequence

Refer to drawing UBCD and to Figure 6-8.

1. When BUS NPR L is asserted, and if none of the disabling conditions are present, the D input to UBCD NPR (1) H becomes high and this flip-flop is set by the first FREE CLK pulse to occur. UBCD NPR (0) H disables the input to UBCD GRANT BR (1) H: no BRs may be granted while an NPR is present. The next clock pulse, 90 ns later, sets UBCD NPG (1) H, which asserts UBCD PROC NPG H on the Unibus, starts the 10-μs NO SACK timeout one-shot and negates UBCD ENAB BR H, which also disables UBCD GRANT BR.

2. When a device receives and accept this NPG, it asserts BUS SACK L and negates BUS NPR L. The first clock pulse to occur after receipt of these signals sets the SACK and clears the NPR flip-flops. The clock pulse after that sets UBCD DSACK (1) H (delayed SACK, 90-ns deskew) and clears NPG. The only arbitration signal now asserted on the Unibus is SACK.

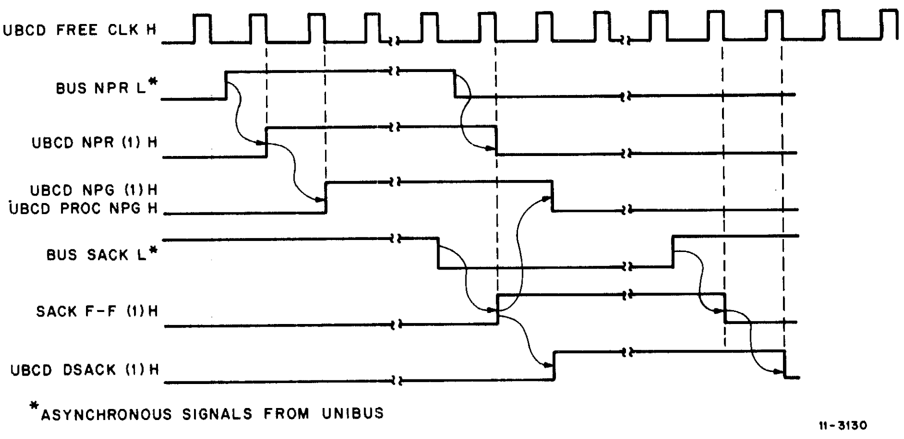


Figure 6-8 NPR-NPG Sequence

3. The device asserts BBSY when the Unibus is free (BBSY negated by previous master), executes its data transfer(s) and negates SACK. The SACK and DSACK flip-flops are cleared by the first and second FREE CLK pulses after receipt of the negation of SACK. If another NPR is pending, PROC NPG H may be asserted 90 ns after DSACK is cleared.

4. If the assertion of BUS SACK L is not received 10 μs after UBCD NPG (1) H is set, UBCD NO SACK (1) H is asserted. This signal forces a sequence similar to that described in (2) above. When UBCD NPG (0) H goes high, UBCD NO SACK (1) H is negated and the SACK and DSACK flip-flops are cleared as in (3) above.

   When the assertion of BUS SACK L is received before the end of the 10 μs timeout, the 74S123 one shot is reset.

6.4.3 BR-BG Interrupt Sequence and Passive Release

The processor checks for both internal and external traps (or interrupts) toward the end of every instruction. This is done by clocking all request lines into the priority flip-flops on TMCA and TMCB [TMCE BRQ STROBE H when RACC UMSC(02:00)=6, at TS3]. If a Unibus request (BUS BR7 L - BUS BR4 L) is asserted, and if its priority is higher than that of any other request present, TMCA HONOR BRn L is asserted (n is the same number as that of the request line being serviced).

The BEN bits of the microprogram cycle, immediately preceding FET.10, always equal 12 and its UADR field, 240. If TMCB BRQ TRUE L is not asserted (no interrupt request), the microprogram branches to FET.10 (instruction fetch). If TMCB BRQ TRUE L is asserted, the microprogram does not branch, but goes to BRK.90 (Flows 12). This cycle does a BEND to cancel the BUST in the previous cycle.

BRK.90 is now entered. If UBCD EXT BRQ H is asserted (= any one of TMCA HONOR BR7 - R4 L asserted) the clock stops in T2, since this is an INTR PAUSE cycle (BSD=1) and UBCD EXT BRQ H is asserted. Refer to drawing TMCA and to Chapter 4, Timing Generator.

Refer to UBCD and to Figure 6-9.

1. TS2 and TMCE INTR PAUSE H gate UBCD EXT BRQ H to the input of UBCD BRQ (1) H. This flip-flop is clocked by FREE CLOCK and its output goes high.

When all NPRs have been serviced and when all grant service is completed, GRANT BR (1) H is set by the first FREE CLK pulse following the one that set BRQ (1) H. This signal gates the TMCA HONOR BRn L signal that is asserted onto the Unibus as UBCD PROC BGn H.

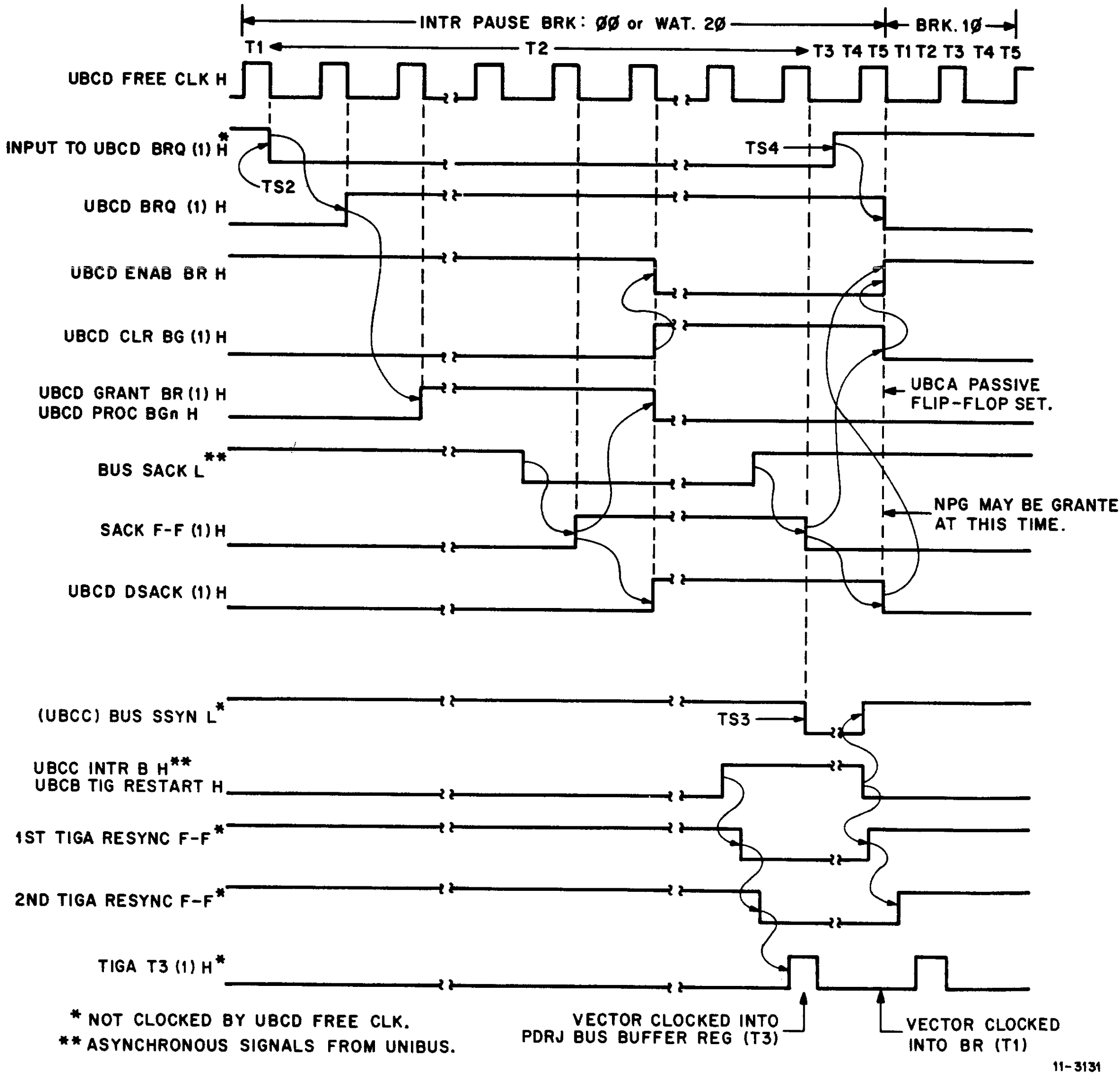


Figure 6-9 INTR Sequence

All NPRs have been serviced if no NPR is pending [NPR (0) H high]. All grant service has been completed if ENAB BR H is high; this is the case when the last NPG is done [NPG (0) H high], the last BG is done and no new grant has been issued [CLR BG (0) H], and DSAK (0) H is high.

2. When the assertion of BUS SACK is received, the SACK flip-flop is set at the first FREE CLK pulse. The next clock pulse sets DSACK (1) H. Since GRANT BR (1) H is high, the same pulse sets CLR BG (1) H, which causes ENAB BR H to be negated. The same clock pulse also clears GRANT BR (1) H by complementing it (J-K flip-flop with both inputs high).

3. The Unibus device now puts the vector on the D lines, asserts INTR and negates SACK.

The first FREE CLK pulse after receipt of the negation of SACK clears the SACK flip-flop; the second clears DSACK.

The assertion of INTR causes UBCB TIG RESTART to be asserted. This signal causes the main clock to be restarted (refer to Chapter 4). A minimum of 75 ns after the assertion of TIG RESTART, T3 is asserted. At this time, the vector is clocked into PDRJ D(15:00) from BUS D(15:00) L. At T4, the BRQ flip-flop is cleared (-TS2 and FREE CLOCK); the next clock pulse clears CLR BG (1) H. ENAB BR H is asserted, and the NPR input to NPG (1) H is enabled. An NPR can now be serviced.

At T1 of the next cycle (=T6 on Flows 12) this data is clocked into the BR (BR←BUS). BUS SSYN L (UBCC) is also asserted by the processor at T3. The device responds to the assertion of SSYN by negating INTR. This, in turn, causes SSYN to be negated, thus ending the INTR Unibus transaction.

4. BUS INTR L also sets UBCC INTRF (1) H. After BRK.00, BRK.10 is executed. TMCB PF(0)*(SF+-TF) H is asserted and TMCB PF(0)*(SF+TF) H is negated at this time, and the branch is to BRK.20(120). Since INTRF is set, TMCB PWRF+INTRF L is asserted and BRK.20 branches to BRK.30 and the Service Flows (SVC.00 - SVC.90) before fetching the first instruction of the subroutine pointed to by the vector.

5. The above is the general case. Passive Release is said to occur when a device that becomes master, by asserting a BR and obtaining a BG, releases the Unibus without doing an INTR. UBCA PASSIVE flags this condition: after a minimum delay of 90 ns, following the receipt by the processor of the negation of SACK, the UBCA flip-flop, whose D input is UBCD CLR BG (1) H, is clocked by the trailing edge of UBCD DSACK; when the device negates BBSY, UBCA PASSIVE L is asserted and restarts the clock via UBCB TIG RESTART H.

BRK.00 is followed as in (4) above by BRK.10 and BRK.20. UBCC INTRF (1) H is not set in this case because BUS INTR L was not received. TMCB PWRF + INTRF L is thus not asserted, BRK.20 branches to RTI.60, and the program resumes at the instruction following that from which the INTR sequence (described above) was entered.

6. The NO SACK logic is the same as that for the NPR-NPG sequence. The negation of DSACK clocks the PASSIVE flip-flop and the sequence in (5) above is followed.

6.5 UNIBUS POWER MONITOR

The processor monitors the condition of all power supplies in the system.

1. Two Unibus signals, BUS ACLO L and BUS DCLO L, inform the processor of the state of the Unibus power supplies: The assertion of BUS ACLO L informs

the processor that the ac power input to a power supply, whose failure might make the bus inoperable, has ceased to be within specifications. The negation of BUS ACLO L informs the processor that all power supplies, whose failure might make the bus inoperable, can maintain dc power within specifications long enough for a complete power-up/power-down sequence.

The assertion of BUS DCLO L informs the procesor that dc power to any bus drivers, receivers or terminators, whose failure would make the system inoperable, is about to fail. The negation of BUS DCLO L informs the processor that dc power to all bus drivers, receivers and terminators, whose failure would make the Unibus inoperable, is within specifications.

2. Two signals from the Cache, ADML MAIN ACLO L and MAIN DCLO L, monitor the Main Memory power supplies. These signals have the same significance and effect as the BUS ACLO and DCLO signals, but are input only to the processor power-up/power-fail circuits, and not to BUS ACLO and BUS DCLO.

   These bus signals are input to the Cache, which performs its power-up initialization sequence upon receipt of the negation of both BUS ACLO and BUS DCLO.

ACLO is always asserted before DCLO; DCLO is always negated before ACLO. Whenever ACLO is asserted, the power supplies must be capable of supplying enough dc power for 5 ms of system operation: this time allows for a 2-ms power-down sequence, plus a 2-ms power-up sequence.

During the power-down sequence, the program stores the contents of volatile registers into core memory; this information is thus preserved during a power failure or power down. It can be retrieved by the power-up sequence, and the program restarted where it was interrupted.

ACLO and DCLO control the power-up and power-down logic shown on drawing UBCE.

6.5.1 Power-Down

Refer to UBCE and to the timing diagram shown in Figure 6-10. When BUS ACLO L is asserted during normal operation, the Power-Down flag, UBCE PDNF (1) H, is set, because UBCE BLOCK DOWN (1) H has been reset at the end of the previous power-up sequence. PDNF is applied to the priority arbitration logic on TMCE; the first BRQ strobe generates TMCE BRQ CLK H, which clocks the interrupt flags into the priority logic. If no higher priority flag is up (CCBJ PARITY TRAP, Memory Management Trap or SL YEL), TMCE HONOR PWRF L is asserted. At the end of the current instruction, the ROM branches to the Service Flows (BRK.90).

At microstate BRK.20, UBCB ACKN B H goes high at TS3 and sets TMCC BLOCK STROBE (1) H. At microstate SVC.90, if no aborts are pending, this signal and TMCE CLK CONF H (BRQ STROBE at T3) generate TMCC AC CLEAR L, which clocks the UBCE PF CLR (1) H flip-flop. This flip-flop is set at this time, since TMCA HONOR PWRF L is asserted. The Q register is cleared to ensure that the first instruction of the power fail routine is executed, in case a request of lower priority than power fail is present.

PF CLR does the following:

1. It asserts TMCA BRQ CLR L, which resets the TMCE priority flip-flops.

2. It resets UBCE PDNF.

3. It starts the 2-ms timer which, at the end of its delay, fires the 1-μs one-shot; the pulse thus generated goes out on the Unibus as BUS DCLO L.

4. It sets UBCE BLOCK DOWN (1) H, which disables the set input to PDNF.

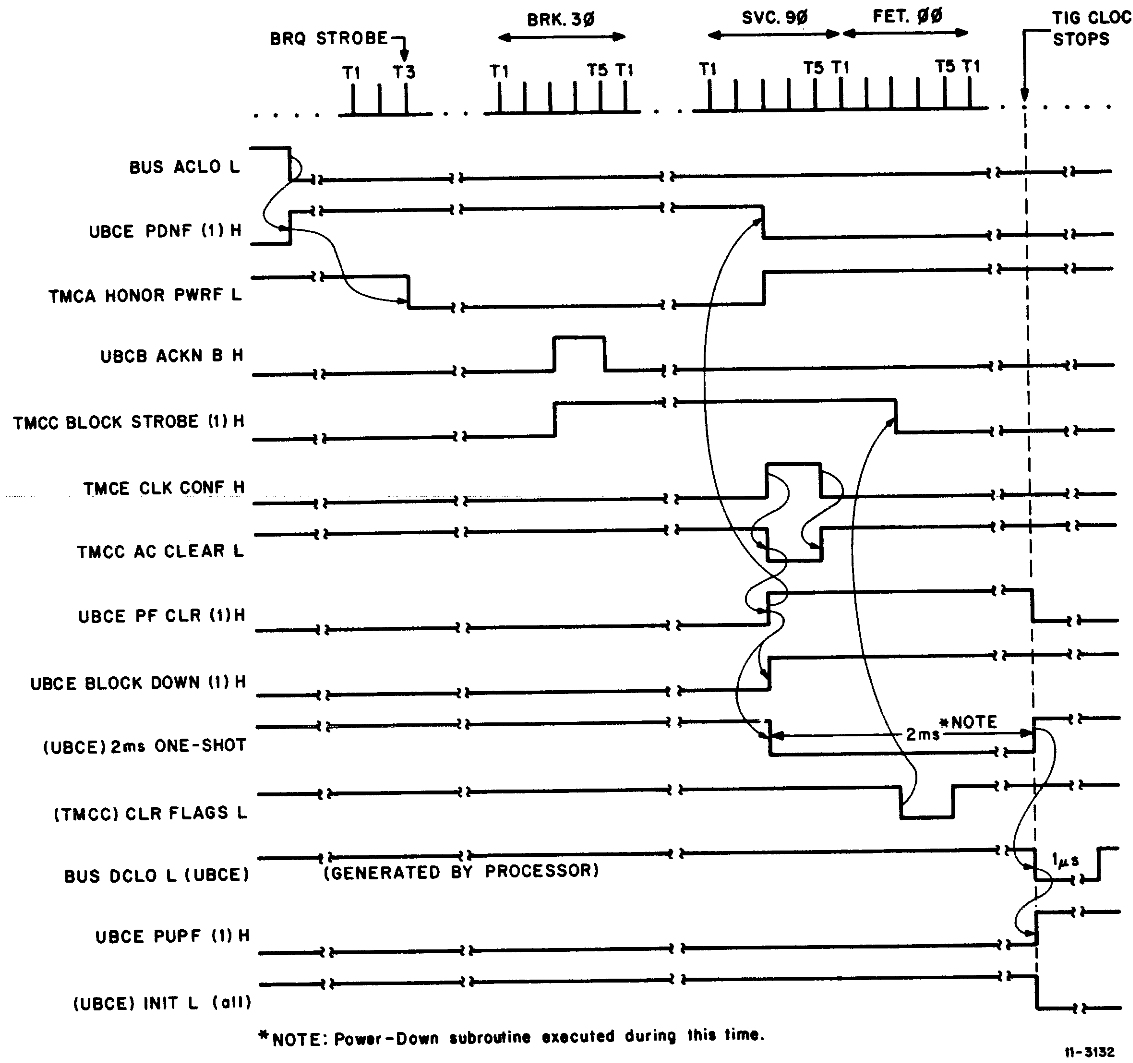


Figure 6-10 Power-Down

By this time, all the internal traps and service routines should have been executed; no further bus transactions can occur, because DCLO asserts the initializing signals:

1. UBCE INT BUS INIT L – clears internal registers PIR, SL, the priority arbitration flip-flops (TMC) and Memory Management.

2. UBCE ROM INIT H – forces the ROM to ZAP.00 (200), and stops and clears the Timing Generator and the Cache timing.

3. UBCE INIT – clears processor, Floating Point Processor, and Cache registers.

4. BUS INIT L – initiaizes Unibus.

In addition, the DCLO generated by the processor sets the UBCE PUPF (power-up) flip-flop, which sets up the power-up sequence, should the DCLO signal not be generated by the power supply, or should ACLO be negated before DCLO is asserted by the power supply.

SL RED During Power Fail
An SL RED abort can only occur during a push to the Kernel stack. Two such pushes are executed during the power fail service routine, in SVC.70 and SVC.90.

If an SL RED error is flagged during one of these pushes, the trap vector is 24 (power fail) but the pushes are made to locations 2 and 0 of the stack, where no SL RED can occur (refer to Paragraph 6.2.2.4, Stack Limit Red). This allows the power fail subroutine to proceed.

1. BLOCK STROBE, STROBE INH, and HONOR PRF are asserted when the abort occurs.
2. TMCC PRIORITY CLR is not asserted because both HONOR PWRF and SL RED are asserted.
3. SL RED sets SERF, which, together with HONOR PWRF, asserts TMCB PWRF L. This signal generates the power-fail vector (24).
4. Since SERF is asserted, SER.00 is entered instead of BRK.80. SER.00 and SER.10 set the Kernel SP to 4.
5. The Service Flows can now be completed by doing the pushes to 2 and 0 without stack error.

6.5.2 Power-Up
Refer to UBCE and to Figure 6-11. When DC power reaches a level at which the logic can operate, but before BUS DCLO L is negated, both UBCE PUPF (1) H and UBCE BLOCK DOWN (1) H are direct-set by DCLO. All INIT signals are asserted by both DCLO and ACLO. While INIT is asserted, the ROM address is forced to 200 (ZAP.00) and the clock is cleared.

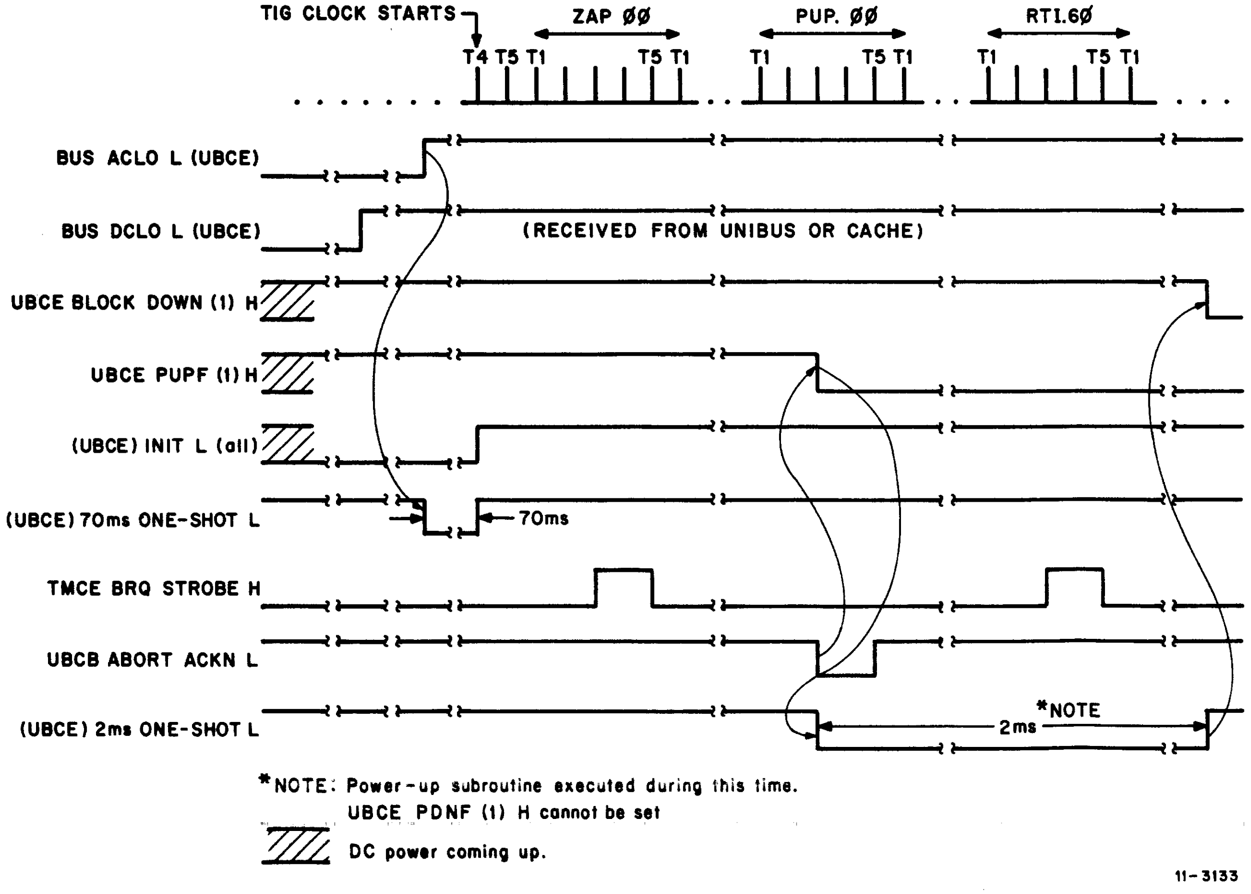


Figure 6-11 Power-Up

As the ac power level rises, BUS DCLO L is negated. When ac power reaches its specified level, BUS ACLO L is negated, UBCE ACLO L goes high and, in conjunction with the assertion of PUPF (1) H, starts the 70-ms timer. During this interval, all INIT signals are asserted and the Cache initializes its tag and data stores.

INIT is negated at the end of the 70-ms delay, the TIG clock is started in T4 (refer to Chapter 4) and the ROM cycles from ZAP.00 – BRK.10 to PUP.00 are executed.

At T3 of PUP.00, UBCB ABORT ACKN L is asserted; this clears PUPF. When PUPF is cleared, UBCE PUPF (0) L initiates a 2-ms delay by triggering the 74123 one-shot. For this period of time, BLOCK DOWN remains set and prevents any BUS ACLO L assertion from setting PDNF. This ensures that the processor will complete the power-up sequence before another power-down is initiated. BLOCK DOWN is reset at the end of the 2-ms delay.

The power-up microprogram sequence (PUP.00 – PUP.40) gets a new PC and PS from the location specified by the start vector (SV). Refer to Paragraph 6.1.

6.5.3 PDP-11/70 System Power Control
Each Main Memory drawer power supply and both processor cabinet power supplies contain an 11086 Power Control Card. The ac power monitor circuits (ACLO and DCLO) are on this card. ACLO and DCLO both have two independent open collector output drivers on each 11086. Refer to the Engineering Print Set for a schematic of this circuit.

Table 6-4 lists the processor cabinet and Main Memory cabinet ACLO and DCLO signals.

6.5.3.1 ACLO Connections – Refer to Figure 6-12. The AC LOW signal from all Main Memory power supplies are wire-ORed and transmitted to the Cache (ADML) on the Main Memory Bus cable. The signal is buffered, renamed (ADML ACLO H) and is one of two inputs to the processor power-up/power-down circuits on UBCE.

Table 6-4
ACLO and DCLO Driver Outputs

Signal Name	Unit	Connector & Pin
	ACLO	
AC LO 1	Upper processor H7420	P/J15-8
AC LO 2	Lower processor H7420	P/J22-8
AC LO 3	Lower processor H7420	P/J22-10
AC LO 4	Upper processor H7420	P/J15-10
AC LOW	Main Memory P/S	P/J6-3
AC LOW	Main Memory P/A	P/J6-8
	DCLO	
DC LO 1	Upper processor H7420	P/J15-12
DC LO 2	Lower processor H7420	P/J22-9
DC LO X	Lower processor H7420	P/J22-12
DC LO Y	Upper processor H7420	P/J15-9
DC LOW	Main Memory P/S	P/J6-1
DC LOW	Main Memory P/S	P/J6-2

The processor power supply AC LO 1, AC LO 2, AC LO 3 and AC LO 4 signals are connected to the Unibus AC LO line (BUS AC LO L) at the backplane. This signal is the other input to the processor power-up/power-down circuits on UBCE, where it is ORed with ADML ACLO.

The output of the OR, UBCE ACLO L, is also input to the Cache power-up circuits (ADMJ).

6.5.3.2 DCLO Connections – Refer to Figure 6-12. There are two separate DCLO lines in the PDP-11/70: BUS DCLO (Unibus) and MAIN DCLO (Main Memory). Two signal lines are required because:

1. The signal level on the Unibus (0 V – 5 V) is different from that on the Main Memory Bus (0 V – 3.5 V), and
2. The impedance of the Unibus (120 ohms) is different from that of the Main Memory Bus (75 ohms).

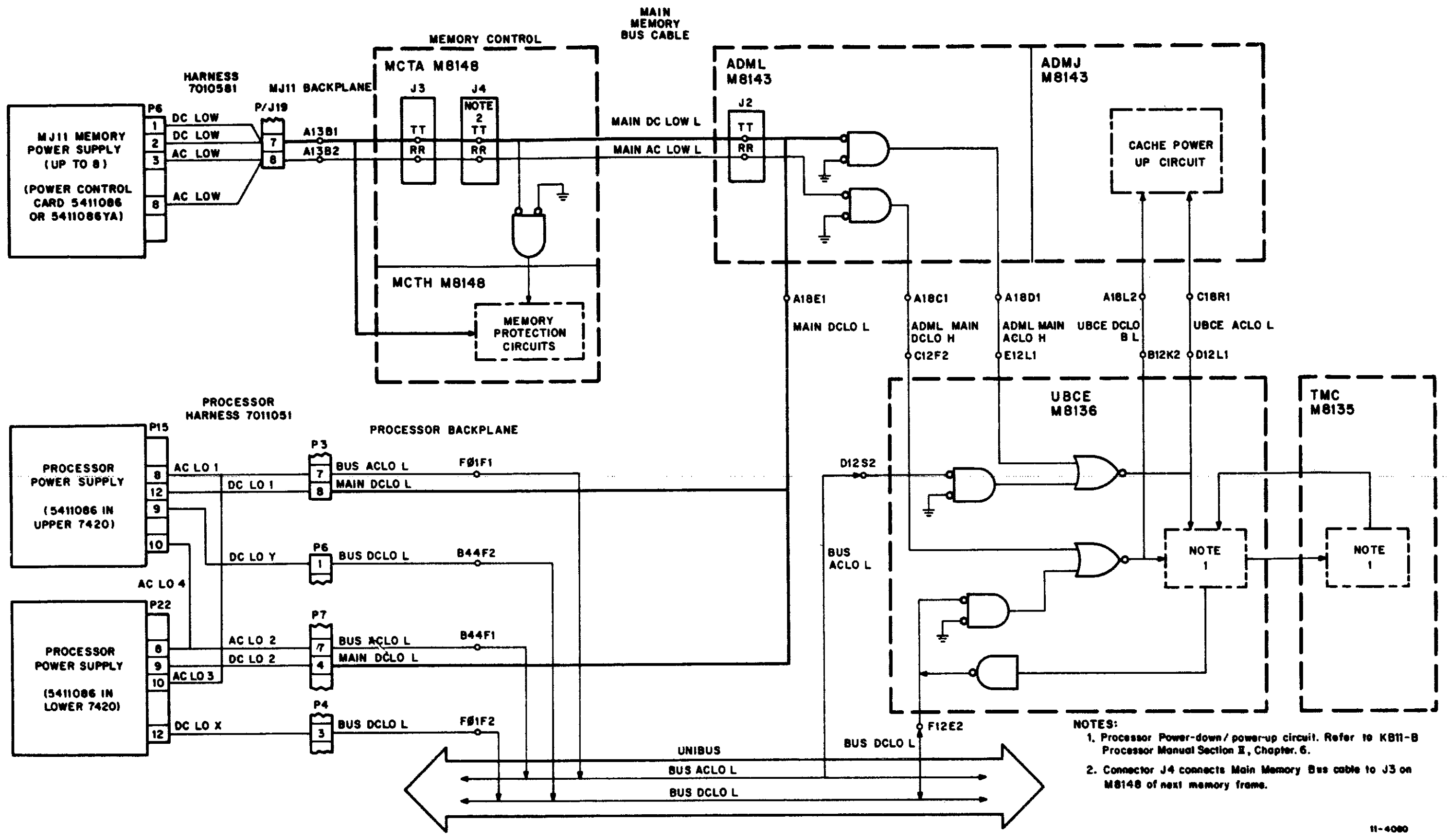


Figure 6-12 PDP-11/70 ACLO and DCLO Connections

NOTES:
1. Processor Power-down/power-up circuit. Refer to KB11-B Processor Manual Section II, Chapter. 6.
2. Connector J4 connects Main Memory Bus cable to J3 on M8148 of next memory frame.

BUS DCLO L is the wire-OR of DC LO Y (upper processor H7420 power supply), DC LO X (lower processor H7420) and the DCLO signals from all devices on the Unibus. It is one of two inputs to the processor power-up/power-down circuits on UBCE.

MAIN DCLO is the wire-OR of DC LO 1 (upper processor H7420), DC LO 2 (lower processor H7420) and both DC LOW outputs from all Main Memory drawers (via the Main Memory Bus cable). MAIN DCLO is buffered in the Cache, renamed (ADML MAIN DCLO H), and is the second input to the processor power-up/power-down circuits.

BUS DCLO and MAIN DCLO are ORed (UBCE) and input to both the Cache power-up and to the processor power-up/power-down circuits. MAIN DCLO, however, is the only input to the Main Memory protection circuitry (MCTH). This circuitry inhibits the memory write operations on power down 3 μs after receipt of DCLO.

6.5.3.3 Power Down – In the PDP-11/70, these interconnections are such that a power failure from any device (Unibus device, processor or Main Memory)

1. Causes the processor to trap to location 24 and to perform the power-down subroutine, and

2. Causes the Cache to prevent all access to Main Memory when DCLO is asserted at the end of the 2-ms power-down subroutine time allotment.

In addition, when the power failure is a processor or a Main Memory failure, the Main Memory protection circuits are activated when MAIN DC LOW is asserted by either the processor or the Main Memory power supplies.

SECTION III

CONSOLE

Unless otherwise indicated, references within this section pertain to this section only.

SECTION III CONSOLE
CONTENTS

CONTENTS (Cont)

ILLUSTRATIONS

TABLES

INTRODUCTION

INTRODUCTION

The PDP-11/70 Console, drawing D-CS-54-11294-0-1, allows direct control of the KB11-C computer system. The Console is used for starting, stopping, resetting and debugging. Its power switch may be used as the master switch for a system. Indicator lights and the other switches provide facilities for monitoring, system control and maintenance operations, during which the KB11-C can be made to execute single instructions or single Unibus or Memory cycles. The contents of any memory location or of any register can be examined, and data can be entered manually from the switches.

Chapter 1 describes the various components of the Console and their use; Chapter 2 describes the logic that controls Console operations.

CHAPTER 1
SWITCHES, INDICATORS AND OPERATION

Refer to Figure 1-1.

1.1 OPERATIONAL SWITCHES

1.1.1 Power and Lamp Test Switches

The POWER switch is a three-position, key-operated switch.

> *OFF* – Causes power to be removed from the switched outlets of the Power Controller. Renders the system inoperative.
>
> *POWER* – Power is applied to the system. All switches are operational.
>
> *LOCK* – Same as POWER, except that the LOAD ADRS, EXAM, DEP, CONT, ENABLE/HALT, S INST/S BUS CYCLE and START switches are disabled. All other switches are operational.

The LAMP TEST switch is the white switch between Switch Register 0 and LOAD ADRS. When raised, it turns all the indicators on. It is used for maintenance.

1.1.2 LOAD ADRS Switch

The LOAD ADRS switch is a momentary action switch. When this switch is depressed, bits 21 – 16 of the Switch Register are loaded into SCCK SWR(21:16) B (1) H, and bits 15 – 00 into the PCA and the SR. The address displayed in the ADDRESS display indicators is a function of the ADDRESS SELECT switch (Paragraph 1.2.1 below).

1.1.3 EXAM Switch

The EXAM(ine) switch is a momentary action switch. When it is depressed, the contents of the location specified by the ADDRESS display is shown by the DATA indicators, if the DATA SELECT switch is in the DATA PATHS position.

The ADDRESS display shows either a virtual or a physical address, as determined by the ADDRESS SELECT switch. Refer to Paragraph 1.2.1.

1.1.4 DEP Switch

The DEP(osit) switch is a momentary action switch. When it is raised, the contents of bits 15 – 00 of the Switch Register are written into the location specified by the physical address generated by the last LOAD ADRS operation. The data written is shown by the DATA indicators if the DATA SELECT switch is in the DATA PATHS position.

The ADDRESS display shows either a virtual or a physical address, as determined by the ADDRESS SELECT switch. Refer to Paragraph 1.2.1.

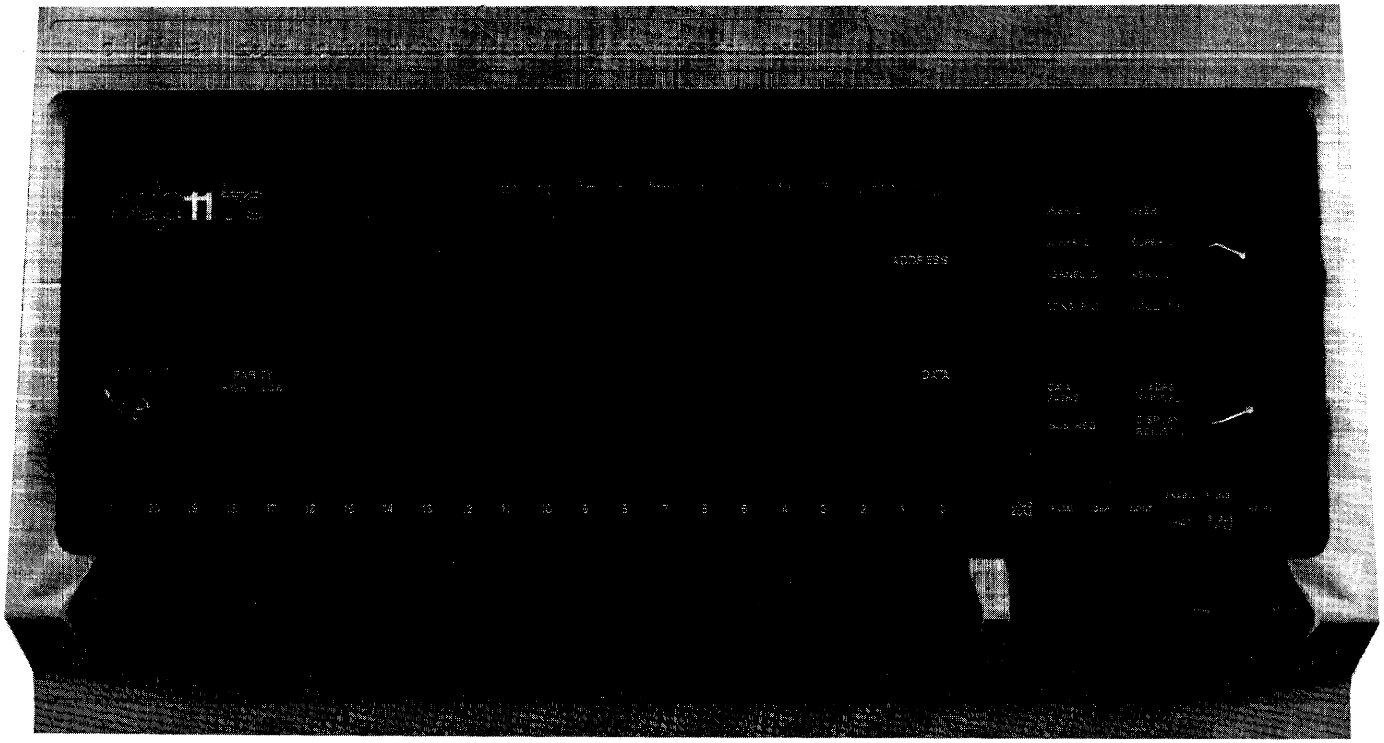
1.1.5 Step Operations

If several consecutive EXAM operations are performed, the address is incremented by 2 for each operation after the first one. Thus, it is possible to examine a series of consecutive word addresses without doing a LOAD ADRS for each EXAM.

In the same manner, it is possible to execute a series of DEP operations without doing a LOAD ADRS for each one.

The following sequence illustrates these operations:

Operation **(Activate Switch)**	**Location Shown in** **ADDRESS Display**
LOAD ADRS	X
EXAM	X
DEP	X
EXAM	X
EXAM	X+2
(Result is EXAM – STEP)	
DEP	X+2
EXAM	X+2



7312-16

Figure 1-1 PDP-11/70 Console

1.1.6 CONT Switch

The CONT(inue) switch is a momentary action switch whose action depends upon the position of the HALT/ENABLE switch:

> *ENABLE* – Resumes program execution at the point where it was stopped by the HALT switch or by a HALT instruction.
>
> *HALT* – Used in conjunction with the S INST/S BUS CYCLE switch. See Paragraph 1.1.8.

The CONT switch has the same effect as the Maintenance Module Stepper Switch, XMAA S4, when executing single ROM cycles or UPB stops, but not when executing single clock cycles.

1.1.7 ENABLE/HALT Switch

The ENABLE/HALT switch is a two-position switch:

> *ENABLE* – Used in conjunction with the START or CONT switches, allows program execution.
>
> *HALT* – Stops program execution.

1.1.8 S INST/S BUS CYCLE Switch

The S(ingle) INST(ruction)/S(ingle) BUS CYCLE switch is used in conjunction with the CONT switch when the HALT/ENABLE switch is in the HALT position:

> *S INST* – When CONT is depressed, a single instruction is executed and the processor stops in CON.00. EXAM and DEP operations may then be executed. The contents of the DATA Display indicators may only be determined by examination of the microprogram Flows for the instruction that has just been executed.
>
> *S BUS CYCLE* – When CONT is depressed, execution is resumed but stops in T5 of PAUSE of the first Unibus or Memory cycle to be executed.
>
> The ADDRESS display then contains the address of the location at which the bus cycle was performed (virtual or physical, depending on the position of the ADDRESS SELECT switch).
>
> If the DATA SELECT switch [illegible] ting BUS REG (Bus Register), the DA[illegible] lay lights, on a read operation, will contain the data that was read (this could be an instruction or data). During a write operation, the lights will contain the data just written (except during a stack operation or Floating Point instruction). LOAD ADRS, EXAM and DEP are disabled in this mode. If an EXAM or DEP operation is desired, the S INST/S BUS CYCLE switch should be changed to S INST and the CONT switch should be depressed once. (This will cause execution until the end of the current instruction). The system will then be ready to perform an EXAM or DEP.

The switch has no effect when the HALT/ENABLE switch is set to ENABLE.

1.1.9 START Switch

The START switch is a momentary action switch whose action depends upon the setting of the HALT/ENABLE switch:

> *ENABLE* – Starts program execution at the address previously loaded by a LOAD ADRS, after resetting the system (INIT).
>
> *HALT* – Resets the system.

The START switch has no effect when the processor is in the RUN state.

1.1.10 Switch Register

The Switch Register consists of the 22 switches labeled 0 through 21. These numbers correspond to the bit positions of their respective switches. The Switch Register is used to manually enter both addresses and data into the KB11-C, and its bits, 15 – 00, may be read under program control; its address is 17 777 570, which is the same as that of the Display Register.

1.2 ADDRESSING AND DATA DISPLAY

1.2.1 ADDRESS SELECT Switch

The ADDRESS SELECT switch is an eight-position rotary switch:

> *VIRTUAL* – Six positions: KERNEL, SUPER and USER I space and KERNEL, SUPER and USER D space. The address displayed is a 16-bit virtual address; bits 21 – 16 are always off.

During Console DEP or EXAM operations, bits 15:00 of the Switch Register are considered to be a Virtual Address. If Memory Management is enabled, this Virtual Address is relocated. The set of PAR/PDRs indicated by the switch position is used.

CONS PHY - (Console Physical). The 22-bit address entered by a LOAD ADRS is the physical address of the Console operation.

PROG PHY - (Program Physical). Displays the 22-bit physical address generated by Memory Management for the current Unibus or Memory cycle.

The ADDRESS SELECT switch indicator lights are driven directly by the switch.

Refer to Paragraph 1.4 which explains the use of the ADDRESS SELECT switch.

1.2.2 ADDRESS Display Indicators

The ADDRESS display indicators show the address of the data deposited or being examined. The address is interpreted as a virtual or physical address in accordance with the position of the ADDRESS SELECT switch. (Paragraph 1.2.1 below).

1.2.3 DATA SELECT Switch

The DATA SELECT switch is a four-position rotary switch:

DATA PATHS - Displays the output of the Shifter. This position is the normal display mode, and is used to show the data examined or deposited by Console operations.

BUS REG - Displays the output of the Bus Register (BR).

μADRS FPP/CPU - Bits 15 - 08 display the current address of the Floating Point Processor microprogram ROM.

Bits 07 - 00 display the current address of the processor microprogram ROM.

DISPLAY REGISTER - Displays the contents of the Light Register. The LR may be written into by using address 17 777 570, which is the same as that of the Switch Register.

1.2.4 DATA Display Indicators

The DATA indicators display the output of the Data Display Multiplexer. The output of the multiplexer is selected by the DATA SELECT switch. Refer to Paragraph 1.2.3.

1.2.5 PARITY Indicators

The PARITY indicators display the parity bits associated with the HIGH and LOW bytes of the word read from Cache Memory. These indicators are off during a write operation.

1.3 EXECUTION INDICATORS

1.3.1 PAR ERR Indicator

The PAR(ity) ERR(or) indicator is on when a Unibus or a memory parity error is flagged.

1.3.2 ADRS ERR Indicator

The ADRS (Address) ERR (Error) indicator is on when an addressing error occurs. Address errors are: non-existent memory, access control violation, page length error, Stack Limit Red, odd address error and Unibus Timeout. This is a dynamic indication of address errors that occur during program execution. It is a static indication during Console functions (i.e., EXAM or DEP).

1.3.3 RUN Indicator

The RUN indicator is on when the processor is executing instructions, but is off during Pause cycles. The RUN indicator is on during a WAIT instruction.

1.3.4 PAUSE Indicator

The PAUSE indicator is on during all Bus Pause and Interrupt Pause cycles, indicating that the processor is waiting for either Memory or a Unibus device.

1.3.5 MASTER Indicator

The MASTER indicator is on either when the processor is Unibus master (UBCA CPBSY) or during Console operations [TMCA CONF (1) L asserted].

1.3.6 KERNEL, SUPER, USER, Indicators

The KERNEL, SUPER and USER indicators show the actual mode in which the processor is operating during each cycle. Refer to Section IV of this manual (Memory Management).

1.3.7 ADDRESSING (Mapping) Indicators

The 16-, 18-, and 22-bit indicators show the Memory Management mapping that is being used during each cycle.

1.3.8 DATA (Space) Indicator

The DATA indicator shows whether I or D space is used during each cycle. It is on when D space is used and off when I space is used.

1.4 USAGE

1.4.1 Memory Reference

Memory references from the Console may be either mapped (i.e., using a virtual address) or unmapped (using a physical address), when Memory Management is enabled. Mapped references are possible only when Memory Management is enabled.

1.4.1.1 Unmapped Reference

1. Set the ADDRESS SELECT switch to CONS PHYS.

2. Enter the 22-bit physical address into the Switch Register.

3. Depress the LOAD ADRS switch. The physical address in shown by the ADDRESS display.

4. Set the DATA SELECT switch to DATA PATHS.

5a. If the EXAM switch is depressed, the contents of the physical memory location entered by the LOAD ADRS operation is displayed by the DATA indicators.

5b. If the DEP switch is raised, the contents of bits 15 – 00 of the Switch Register are written into the physical memory location entered by the LOAD ADRS operation.. This same data is displayed by the DATA indicators.

1.4.1.2 Mapped Reference

1. Set the ADDRESS SELECT switch to one of the virtual positions (refer to Paragraph 1.2.1).

2. Enter the 16-bit virtual address into the Switch Register.

3. Depress the LOAD ADRS switch. The virtual address is shown by the ADDRESS display. Bits 21 - 16 are off.

4. Set the DATA SELECT switch to DATA PATHS.

5a. If the EXAM switch is depressed, the virtual address loaded by the LOAD ADRS operation is relocated by Memory Management. Memory Management (if it is enabled) uses the mapping shown by the ADDRESSING indicators (Paragraph 1.3.8) and the PAR/PDR pair is selected by the ADDRESS SELECT switch. The contents of this address are read and displayed by the DATA indicators.

5b. If the DEP switch is raised, the virtual address is relocated as in the EXAM operation. The contents of the Switch Register are written into the physical memory location pointed to by the physical address. The new contents of this location are displayed by the DATA indicators.

6. If the ADDRESS SELECT switch is now turned to PROG PHY, the physical address corresponding to the virtual address used during the EXAM or DEP operation is displayed by the ADDRESS indicators.

1.4.2 General Register Reference

EXAM and DEP references to the processor General Registers may be executed by entering the address of the register (see Table 1-1) into the SWITCH REGISTER, depressing LOAD ADRS, and then EXAM or DEP, as required. The ADDRESS SELECT switch setting is ignored; mapping to a General Register is not possible.

EXAM-STEP and DEP-STEP operations can be performed on the General Registers, in a manner similar to that for memory locations, except that:

1. ADDRESS display is incremented by 1 (instead of 2).

2. The STEP after address 17 777 717 is 17 777 700, such that the addresses are looped.

3. It is not possible to STEP up to the first General Register (17 777 700) from 17 777 676.

Table 1-1
General Register Addresses

SET 0	
Register 0	17 777 700
.	.
.	.
Register 5	17 777 705
Register, 6 Kernel	17 777 707
Program Counter	17 777 707
SET 1	
Register 0	17 777 710
.	.
.	.
Register 5	17 777 715
Register 6, Super	17 777 716
Register 6, User	17 777 717

CHAPTER 2
LOGIC DESCRIPTION

The Console assembly consists of a printed circuit board, drawing D-CS-5411294-0-1 (KNLA – KNLD), an indicator panel, drawing D-IA-7413126-0-0, and a bezel, E-IA-7409306-0-0. This assembly is mounted on the front of the processor mounting box. It is connected to the PDP-11/70 by the power harness, whose P1 plug connects to the Console J4 connector, and by three flat ribbon cables. One of these connects J1 on the Console to J1 on the M8134 module (PDRH). Another connects J2 on the Console to J1 on the M8140 module (SCCJ). The third cable connects J3 on the Console to J2 on the M8140.

This chapter describes the Console Power Connector and the logic that controls the Console Switches (Paragraphs 2.2 through 2.9) and the Displays (Paragraphs 2.10 through 2.12), in that order.

2.1 POWER CONNECTOR J4 (KNLA)

J4 connects to the Power Harness. It consists of the following lines: +5 VA, which powers the light emitting diode (LED) indicators: GND A which is the return for the LAMP TEST switch (KNLD LAMP TEST L); and with the Power Controller GND IN and GND OUT (refer to Paragraph 2.2).

2.2 POWER SWITCH S31 (KNLA)

The Power Switch controls power to the system through the GND OUT/GND IN connections to the Power Controller, and enables/disables switches S24 – S30 (LOAD ADRS, EXAM, DEP, CONT, ENABLE/HALT, S INST/S BUS CYCLE and START).

Power Controller – Pins 3 and 4 of J4, GND IN and GND OUT go to the Power Controller by way of the power harness. When there is no connection between these two pins, power is removed from the switched outlets of the controller. When the pins are connected, power is applied to these outlets. GND IN and GND OUT are not connected when the power switch is in the OFF position; they are connected when the switch is in the ON (terminals 9 and 10) or in the LOCK positions (terminals 11 and 12).

S24 – S30 (KNLC) use the KNLA SWITCHED GROUND from the power switch; there is no ground connection in the LOCK position, and these switches are then disabled. In addition, KNLA PNL LOCK L is also generated in this position, and forces the HALT/ENABLE switch output to the ENABLE logic value (low). Thus, when the power switch is in the LOCK position, the processor is enabled, the HALT switch is inoperative, and the other switches are disabled, since they cannot be used when the **KB11-C** is running. KNLA SWITCHED GROUND is connected to GND B when S31 is in the OFF position (terminals 2 and 3) and in the ON position (terminals 4 and 5). KNLA PNL LOCK H is brought out to the KB11-C backplane by the SCC module, but is not used by any other part of the processor.

2.3 S1 – S22: SWITCH REGISTER

The Switch Register, S1 – S22 [KNLC SWR(21:00) H], is transmitted from KNLC J3 to J2 of the M8140 module (SCCJ), where it becomes SCCJ SWR(21:00) H. It is read by the processor on the Internal Bus from the multiplexer on SCCH.

2.4 S24 – S30 ("LOAD ADRS" – "START")

S24 – S30 are input to latches (KNLC) for bounce suppression and transmitted from J3 of the Console to J2 of SCCJ (SCCJ CONT SW H – SCCJ HALT SW H). CONT, SINGLE CYCLE, LOAD ADRS, START and HALT are buffered on SCCJ.

SCCJ EXAM SW H and DEP SW H are gated with SCCF GEN RG (1) H and (0) H to generate SCCF REG EXAM H and REG DEP H when a General Register address has been decoded during a LOAD ADRS operation. When any address other than a General Register address is detected, SCCF GEN REG (0) H is high and SCCF EXAM H or DEP H are generated.

The signals derived from S24 – S30, with the exception of ENABLE/HALT (S28), clock the flip-flop shown on UBCF. When any of these switches is actuated, and if the Console Flag is asserted, UBCF CNSL ACT is set at TS3.

These flip-flops are reset at T4 when BCT=2 (CNSL ACKN), at T2 when BSD=1 (ITR PAUSE), or by INIT.

When the processor is halted, it cycles in the CON.00 microprogram state.

When any of the LOAD ADRS, EXAM, DEP, CONT or START switches are activated, a microprogram branch from CON.00 occurs.

2.5 CONSOLE BRANCH

The Console microprogram flows are shown on Flows 14.

2.5.1 Idle State

CON.00 is the KB11-C idle state which is entered upon a HALT. This cycle loops upon itself until one of the Console switches sets UBCF CNSL ACT (1) H. This function is low during the idle state; the Branch Enable field of CON.00 is 14, making both RACK BEF(3:2)3 H and RACK BEF(1:0)0 H high. RACK BRCAB06 L is thus asserted. Since the UADR field of CON.00 is 070, bit 6 is forced to 1, the ROM address [RACL RADR(07:00) H] becomes 10, the address of CON.00, which thus succeeds itself. RACK BRCAB06 L is not used by any other microstate.

2.5.2 LOAD ADRS

If the LOAD ADRS switch is now depressed, UBCF CNSL07 (1) H and UBCF CNSL ACT (1) H are both asserted; this causes RACK BRCAB07 to be asserted, thus generating a ROM address of 270 (ADR.00). [UBCF CNSL07 (0) H forces UBCH CNS(02:00) H low]. RACK BRCAB07 L is used only by LOAD ADDRESS.

2.5.3 RACK BRCAB(02:00) L

These bits determine the branch required by the eight remaining functions: START, CONT, EXAM, DEP, STEP EXAM, STEP DEP, REG EXAM/DEP and REG EXAM/DEP STEP. The bits are shown on UBCH.

2.5.4 START and CONT

START and CONT are encoded on UBCH. Since BCF CNSL07 (0) H is high (LOAD ADRS is not depressed), UBCH CNSL(02:00) equals 6 for START and 7 for CONT. Since the processor is in the idle state, RACK BRCAB(02:00) L force the next cycle microaddress respectively to 076 or 077.

2.5.5 EXAM and DEP Switches

As described in Chapter 1, every successive depression of the EXAM switch after the first one causes the address to be incremented, thus making it possible to examine successive locations without reloading the address. This same procedure is followed for DEP, or when operating on General Registers. Operations following the first one are called STEP operations. Refer to Flows 14.

The logic shown on UBCH stores UBCH CNSL(02:00) H and UBCH MSB DATA L, in the 74S175 register, the output of which is decoded by the 7442S. The functions generated by the outputs of these decoders are gated with the outputs of the UBCH switch flip-flops and thus generate a modified UBCH CNSL(02:00) H value, which in turn causes a different branch address to be generated when EXAM or DEP are depressed more than once. Note that when R3(1) of the 74S175 is high, the lower 7442 decoder is disabled (no outputs f0 – f7 can be true), while the upper 7442 is enabled, since R3(0) is low; if R3 is reset, the opposite is true.

Register operations are similar to Memory operations. The branch after CON.00 determines whether the operation is or is not a STEP operation. A second branch after this executes either an EXAM or a DEP.

Figure 2-1 shows a sequence of operations, shown above the waveshapes, the condition of the various modifying functions, and the inputs to the RACK logic.

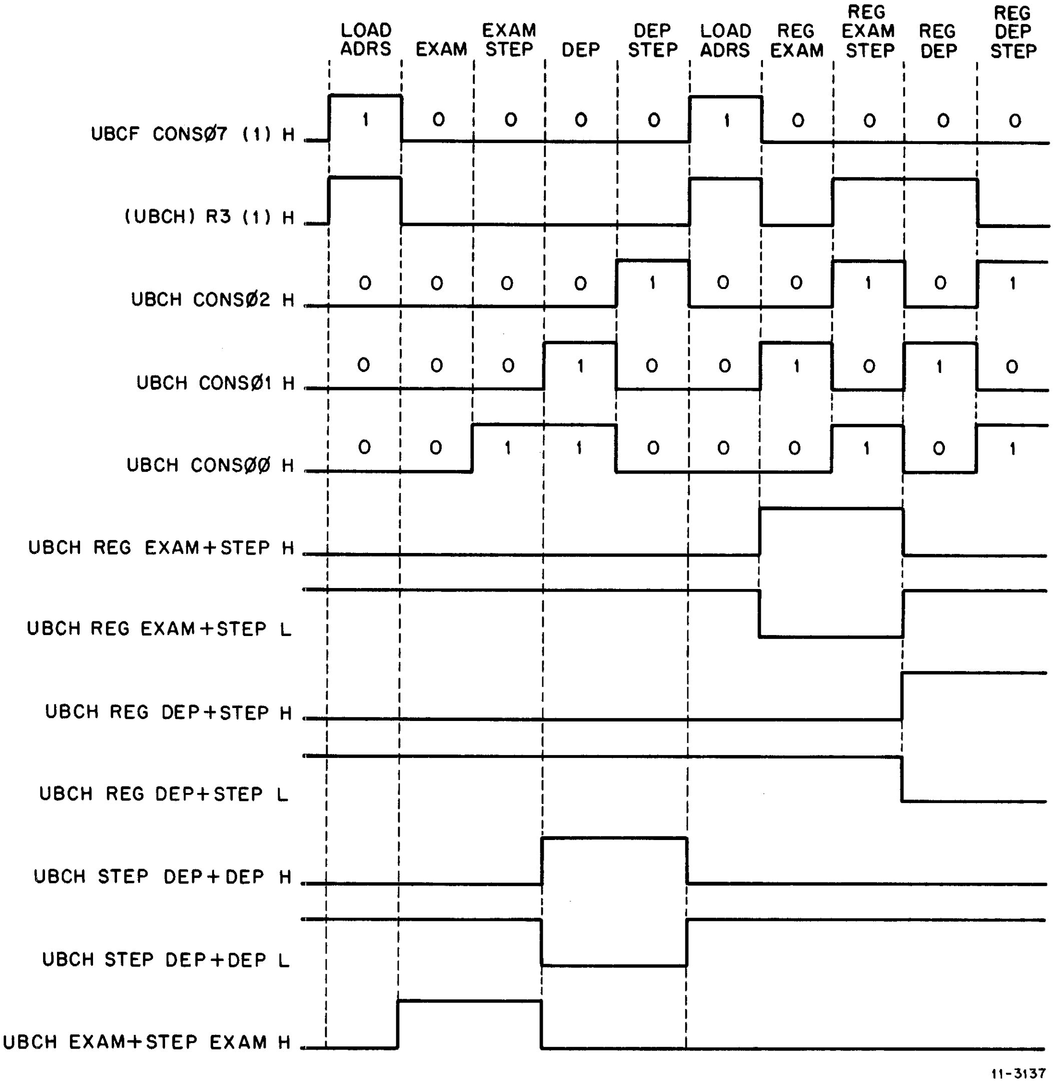


Figure 2-1 Step Branch Address Modification

2.6 ENABLE/HALT SWITCH IN HALT POSITION

Paragraphs 2.6.1 through 2.6.4 describe the effect of the operational switches when the ENABLE/HALT switch is in HALT. When this is the case, KNLC HALT SW H and SCF HALT H are high.

2.6.1 Single Instruction

If the S INST/S BUS CYCLE switch is in the S INST position, KNLC SINGLE BUS CYCLE SW H and SCCF SINGLE CYCLE H are low, and UBCF STOP L is asserted. At the next BRQ strobe (MSC=6), TCE CLK CONF H is asserted at T3 and sets the Console Flag [TMCA CONF (1) H]. TMCB BRQ TRUE is then asserted and the instruction currently being executed branches (when completed) to BRK.90 (refer to Flows 12) on a microprogram cycle where BEN=12 and UAD=240. BRK.00 follows BRK.90, and its BEN bits=12, with UAD=130. Since the Console Flag is set (CONF), the next microprogram state is CON.00 (Flows 14), in which the processor cycles until Console action is initiated by the operator.

2.6.2 Continue

If the CONT switch is now depressed, CON.10 is entered, followed by BRK.10 and BRK.20 (Flows 12). Since neither BUS INTR nor power-down (TMCA HONOR PWRF L), nor an internal trap has caused entry into the BRK sequence, UBCC (PWRF+INTR) L is not asserted and a branch is made to RTI.60 (Flows 2). During this cycle, the Console Flag is cleared during TS3 by UBCH CLR CONF L [BCT=2, or CONS.ACKN and UBCF CONT (1) H]. The CONT switch flip-flop [UBCF CONT (1) H] is cleared at T4 by UBCF ACKN T4 (BCT=2 and TS4).

The instruction following the one at which the processor stopped is now fetched (FET.00) and executed; since the ENABLE/HALT switch is still in the HALT position, the Console Flag is again set by the BRQ strobe and the processor stops after executing one instruction.

2.6.3 Single Bus Cycle

If the S INST/S BUS CYCLE switch is in the S BUS CYCLE position, the processor stops in T5 of the current Unibus or Cache cycle. Refer to Section II, Chapter 4 (Paragraph 4.9) of this manual. NPRs are not allowed when the switch is in this position (UBCF DISABLE NPR L).

2.6.4 Console Reset

If the START switch is depressed when the ENABLE/HALT switch is in the HALT position, UBCF CNSL RESET L is asserted. This signal generates all three INIT signals and sets the Console Flag.

2.7 ENABLE/HALT SWITCH IN ENABLE POSITION

Paragraphs 2.7.1 through 2.7.3 describe the effect of the operational switches if the ENABLE/HALT switch is put into the ENABLE position. When this is the case, KNLC HALT SW H and SCCF HALT H go low.

2.7.1 Continue

When the processor is halted and the CONT switch depressed, the sequence is similar to that described in Paragraph 2.6.2. The Console Flag, however, is not set at the end of the first instruction, and program execution continues instead of stopping.

2.7.2 Single Bus Cycle

The SINGLE BUS CYCLE switch is disabled when the HALT/ENABLE switch is in the ENABLE position.

2.7.3 Console Start

UBCF START (1) H can only be set if the Console Flag has previously been set. START asserts UBCF STATUS CLR L which sets the processor mode bits [PS(15:14)] to 00 or Kernel. The START switch signal, UBCF START L, clocks SCCF HALT H into a flip-flop on UBCE; this flip-flop sets if the HALT/ENABLE switch is in the ENABLE position.

The KST.00 (Flows 14), RES.00 and RES.10 (Flows 3) cycles are then executed.

In RES.10, BCT=4 (INIT if Kernel Mode). Since PS(15:14) have been set to 00 by UBCF STATUS CLR L, UBCC START INIT (1) H is set at T3. This function:

1. direct-sets UBCC RIP+FPSYNC H and
2. starts the 100 μs UBCC RESET WAIT one-shot.

RES.20 is now executed, and the microprogram cycles in this state until RIP+FPSYNC H is negated.

1. RESET WAIT is still on. When it goes off, at the end of the 100 μs, the RESET ABORT (1 μs) and UBCC RESET (1) H (10 ms) one-shots are started.
2. RESET (1) H clears UBCC START INIT (1) H and keeps RIP+FPSYNC H asserted.
3. RESET (1) H is ANDed with the flip-flop on UBCE that was set by the START switch. This asserts UBCE START INIT L, which in turn asserts all the INIT signals with the exception of ROM INIT.

4. At the end of 10 ms, RESET (1) H goes low and INIT is negated. RESET (0) H goes high and a T3 RIP+FPSYNC H is also negated. This causes a branch to FET.03 instead of to RES.20 at the end of the cycle (BEN=10, UADR=334), and the instruction whose address is displayed is fetched and executed.

5. The BUST in FET.03 clears (at T3) the flip-flop on UBCE that was set by the START switch.

2.8 LOAD ADDRESS

During CON.00, bits 15 - 00 of the Switch Register are loaded into the BR. During ADR.00 (LOAD ADDRESS), the contents of the BR are loaded into the SR and into the PCA. These bits are used in any subsequent Console operation other than a LOAD ADRS.

The actual physical address used during these operations is determined by Memory Management from the position of the ADDRESS SELECT switch.

2.9 EXAM AND DEP OPERATIONS

EXAM, DEP, REG EXAM/DEP and their respective STEP operations are described by Flows 14.

2.10 ADDRESS DISPLAY

The ADDRESS DISPLAY indicators are driven by KNLB VA(03:00) and KNLB DISP ADRS(21:04) H. These signals are received on J2 by the Console. They originate on the M8140 module (SCCJ) connector J1. SCCA VA(03:00) H, SCCF DISP ADRS(05:04) H and SCCK DISP ADRS(21:06) H are the sources for the KNLB signals.

Refer to Table 2-1. The address displayed depends on whether or not it is a General Register (GR) address (17 777 700 - 17 777 717).

2.10.1 General Register (GR) Address

If the address is a GR address, bits 00:03 display the register number (0 to 17), bits 4 and 5 are 0s (off), and bits 06:21 are 1s (on).

SCCF GEN REG ADRS is asserted (Switch Register bits 21 - 06 high, bits 05 and 04 low) and SCCF GEN REG (1) H is set when the LOAD ADDRESS switch is depressed. This forces SCCF DISP ADRS(05:04) low and their corresponding indicators off, and also forces low both select inputs to the SCCK DISP ADRS(21:16) H multiplexer, thus selecting its A inputs (+3 V) and forcing the corresponding indicators on. The SCCK DISP ADRS(15:06) H multiplexer is disabled by SCCF GEN REG (1) H and its outputs are high, thus forcing their corresponding indicators on. VA(03:00) determine the state of address indicators 03–00.

2.10.2 Memory Address

If the address is not a GR address, the address display is a function of the ADDRESS SELECT switch, described in Paragraph 1.12. The output of this switch is encoded on the Console board. Three signals, NLD DISP ADRS SEL(2:0) H are thus generated. They are decoded on SSRK and used in the Memory Management logic. Two of these signals control the multiplexers on SCCK and determine the source of the address display, as shown in Table 2-1. VA(05:00) are used for all three mappings, since these bits never change (they are not relocated). VA(15:06) is used for the VIRTUAL

Table 2-1
Address Display

Display Indicators	Virtual (6 positions)	Address Select Switch CONS PHY	PROG PHY	General Register Address
00–03	VA(00:03)	VA(00:03)	VA(00:03)	VA(00:03)
04,05	VA04,05	VA04,05	VA04,05	OFF
06–15	VA(06:15)	VA(06:15)	VA(06:15)	ON
16–21	OFF	SWR(16:21)	PA(16:21)	ON

and CONS PHYS positions (the Switch Register is loaded into the SR after a LOAD ADRS and read from the BAMX). In VIRTUAL, bits 21:16 are forced off. In CONS PHY, SCCK SWR(21:16) H are read. I PROG PHY, PA(21:06) are displayed.

2.11 DATA DISPLAY

The DATA indicators [KNLA DISP D(15:00) H and DISP PAR HI (and LO) H] receive their input from the Data Display multiplexer, PDRF DISP D(15:00) H, and from two flip-flops, PDRH IND HI (or LO) PAR H.

PDRF DISP D(15:00) H selects one of four inputs. (Refer to Paragraph 1.20.) The select inputs to this multiplexer are encoded from the DATA SELECT switch [KNLD DISP DATA SELL (or SEL0) H] and input to S1 and S0 of the multiplexer (PDRF DISPS1 L and DISPS0 L) after being inverted.

The PARITY indicators receive their input from the parity flip-flops on PDRH. The Cache parity bits, DTML HI (or LO) BYTE PAR H are clocked into the same flip-flop IC as PDRB BR(15:12)A H. The output of these flip-flops, PDRB HI (or LO) PAR H are clocked into PDRH DISP HI (or LO) PAR by UBCA IND CLK H. This signal is asserted at T4 during the ROM state following the Pause cycle of all Cache DATI/P cycles. The indicators are cleared at T4 of PAUSE of all Unibus cycles or Cache DATO/B cycles by UBCB CLR IND (0) H.

2.12 MISCELLANEOUS INDICATOR LOGIC

The Console indicators not described in Paragraphs 2.10 and 2.11 are driven by the logic signals listed below (in the same order as they appear in Chapter 1).

ADDRESS SELECT SWITCH (1.2.1) - The indicators are driven directly by the switch.

DATA SELECT SWITCH (1.2.3) - The indicators are driven directly by the switch.

PARITY (1.2.5) - PDRH IND HI PAR H and LO PAR H.

PAR ERR (1.3.1) - UBCB IND PAR ERR H

ADRS ERR (1.3.2) - SCCF IND ADRS ERR H

RUN (1.3.3) - TMCF IND RUN H.

PAUSE (1.3.4) - TMC IND PAUSE H.

MASTER (1.3.5) - UBCF IND MASTER H.

KERNEL, SUPER, USER (1.3.6) - Driven by a decode (on the Console board) of SSRB MMR0 MODE 0 H and MMR0 MODE 1 H.

ADDRESSING (Mapping) (1.3.7) - SCCF IND 16 (or 18 or 22) BIT MODE H.

DATA (Space) (1.3.8) - SAPK IND DATA H.

SECTION IV

MEMORY MANAGEMENT

Unless otherwise indicated, references within this section pertain to this section only.

SECTION IV MEMORY MANAGEMENT
CONTENTS

Page

SECTION IV MEMORY MANAGEMENT
CONTENTS (Cont)

Page

ILLUSTRATIONS

SECTION IV MEMORY MANAGEMENT
ILLUSTRATIONS (Cont)

TABLES

INTRODUCTION
PDP-11/70 ADDRESS SPACE

Processor-generated addresses differ from those that address memory; thus, the processor addresses are termed Virtual Addresses (VA), and the memory addresses are termed Physical Addresses (PA).

The VA is that generated by the program. It consists of 16 bits. The PA is the result of modifying the VA in 18- or 22-bit mapping. It is the address sent to the Cache (22 bits) or to the Unibus (18 bits). Three separate address spaces are used:

1. 16 bits, program virtual space
2. 18 bits, Unibus space
3. 22 bits, physical space

The KB11-C Processor System generates a 22-bit address, which allows a possible address space of 2048K words (2^{21} = 2,097,152). Addresses 00 000 000 - 17 777 777 can be used; they are called Physical Addresses.

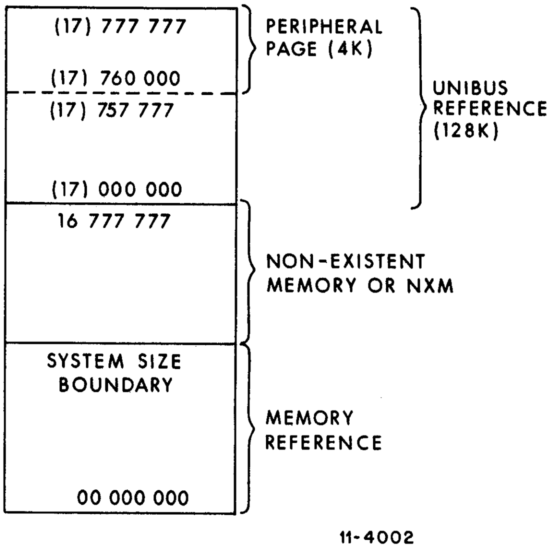


Figure I-1 Physical Address Space

Refer to Figure I-1, which shows the components of the PA Space.

1. Unibus Reference includes 128K PAs, 17 000 000 - 17 777 777, which correspond to Unibus addresses 000 000 - 777 777. The Unibus reference in turn includes the following:
   a. The Peripheral Page, which is reserved for Unibus device registers; it consists of 4K PAs, 17 760 000 - 17 777 777 (Unibus addresses 760 000 - 777 777).
   b. The remaining 124K addresses, 17 000 000 - 17 757 777 (Unibus addresses 000 000 - 757 777) may be used by Unibus devices to access memory.

2. Memory Reference includes PAs from 00 000 000 through the system size boundary, which is the highest address available in the system Main Memory. There may be no discontinuity in Main Memory, i.e., available memory locations must be numbered sequentially - from 00 000 000 through the system size boundary. The highest possible address is 16 777 777. Maximum possible memory is 1920K words (2^{21} - 2^{17} = 1,966,080, or 2048K - 128K = 1920K).

3. Non-Existent Memory or NXM includes the PAs from the system size boundary plus 1 - 16 777 777.

ADDRESS RELOCATION

The PDP-11/70, like all other PDP-11s, generates a 16-bit Virtual Address in the range of 000 000 – 177 777. In order to access the Unibus, which requires an 18-bit address, and Main Memory, which uses a 22-bit address, the VA must be relocated. In the same manner, Unibus devices generate an 18-bit address, which must be expanded to 22-bits in order to access Main Memory.

Refer to Figure I-2. The 16-bit VA is expanded to a 22-bit PA by Memory Management. If the four high-order bits of this PA are all 1s (bits 21:18), the Unibus is referenced. If these four bits are not all 1s (addresses 00 000 000 – 16 777 777), Main Memory is referenced.

The Unibus Map performs a function similar to that of Memory Management: it expands Unibus addresses from 18 to 22 bits. This function is also called "mapping." The Map accepts Unibus addresses 000 000 – 757 777 and relocates them to the PA space (00 000 000 – 16 777 777).

Relocation is controlled by the program, which can enable or disable the Unibus Map and/or Memory Management. The program also specifies the manner in which the addresses are modified when these devices are enabled.

MEMORY MANAGEMENT MAPPING

Three methods of mapping are available to Memory Management:

1. 16-bit mapping, when MMR0 is cleared. (Refer to Figure I-3.)

2. 18-bit mapping, when bit 4 of MMR3 is cleared and bit 0 of MMR0 is set. (Refer to Figure I-4.)

3. 22-bit mapping, when both bit 4 of MMR3 and bit 0 of MMR0 are set. (Refer to Figure I-5.)

MMR0 responds to Unibus address 17 777 572, MMR3 to address 17 772 516.

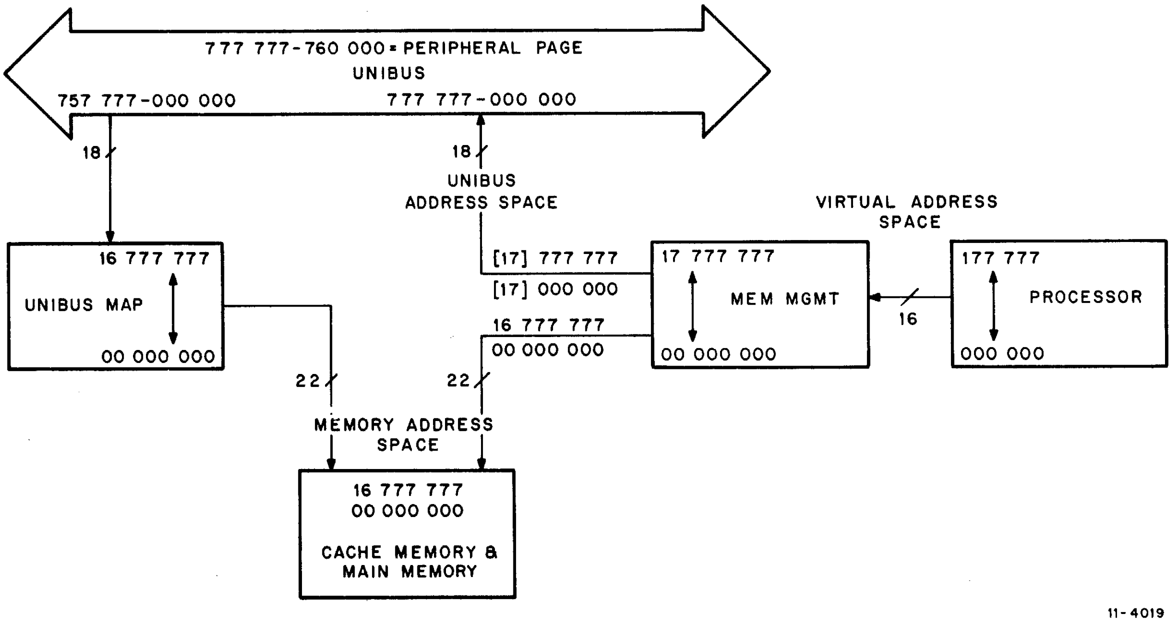
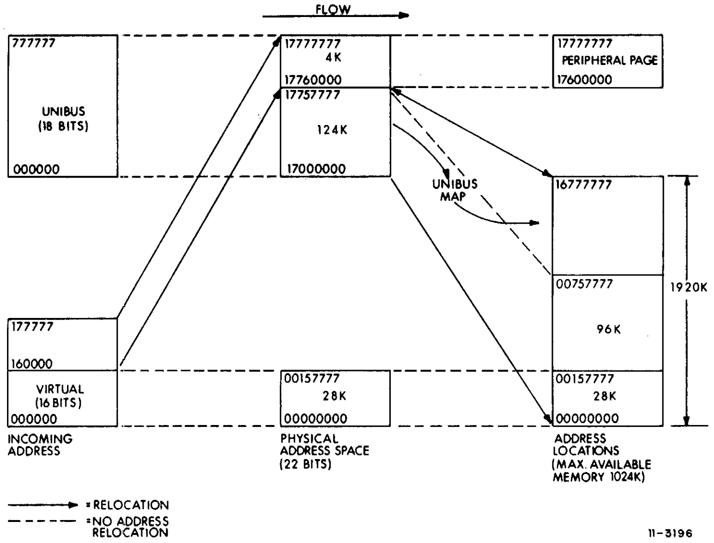
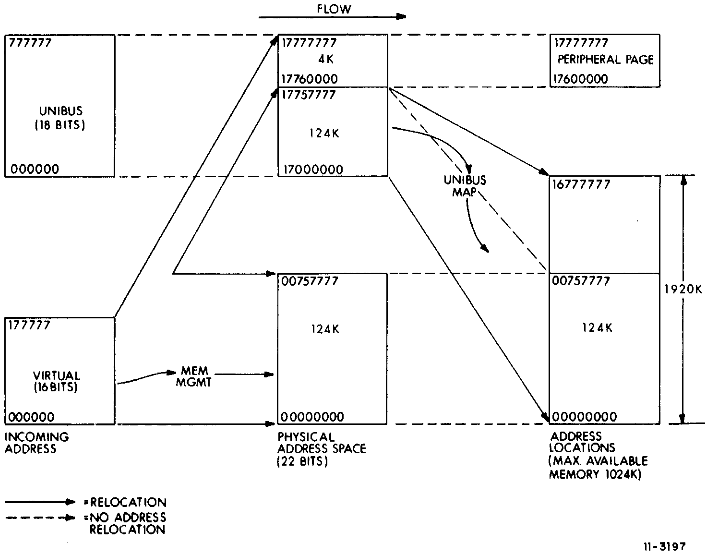


Figure I-2 11/70 Address Space



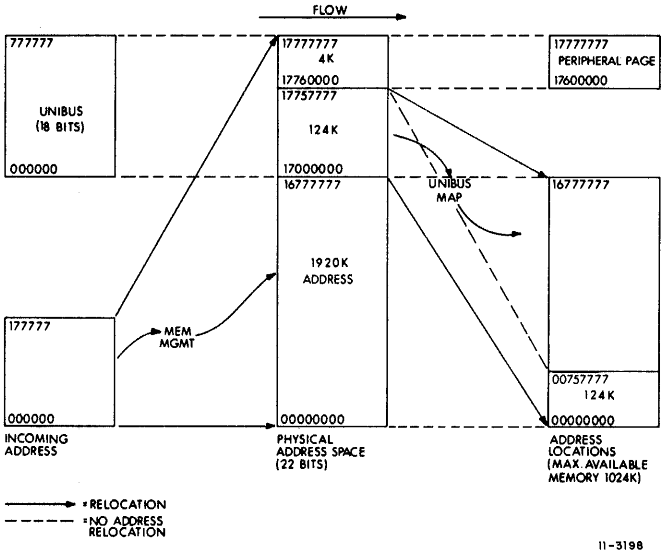
In 16-bit mapping, only addresses in the ranges of 17 760 000 - 17 777 777 (Peripheral Page) and of 00 000 000 - 00 157 777 (Main Memory) may be generated. In 16-bit mapping, the PDP-11/70 operates as the PDP-11/20, or as the PDP-11/45 with Memory Management disabled.

Figure I-3 16-Bit Mapping



In 18-bit mapping, only addresses in the ranges of 17 760 000 - 17 777 777 (Peripheral Page) and of 00 000 000 - 00 757 777 (Main Memory) may be generated. In 18-bit mapping, the PDP-11/70 operates as the PDP-11/45, with Memory Management enabled.

Figure I-4 18-Bit Mapping



In 22-bit mapping, the VA may be relocated to any address in the PA space (00 000 000 - 17 777 777). This is the only mapping in which PAs 17 000 000 - 17 757 777 can be generated; they correspond to Unibus addresses 000 000 - 757 777, or the 124K Unibus locations which are not reserved for the Peripheral Page. The addresses in this range can be useα by the program to access Main Memory via the Unibus Map.

Figure I-5 22-Bit Mapping

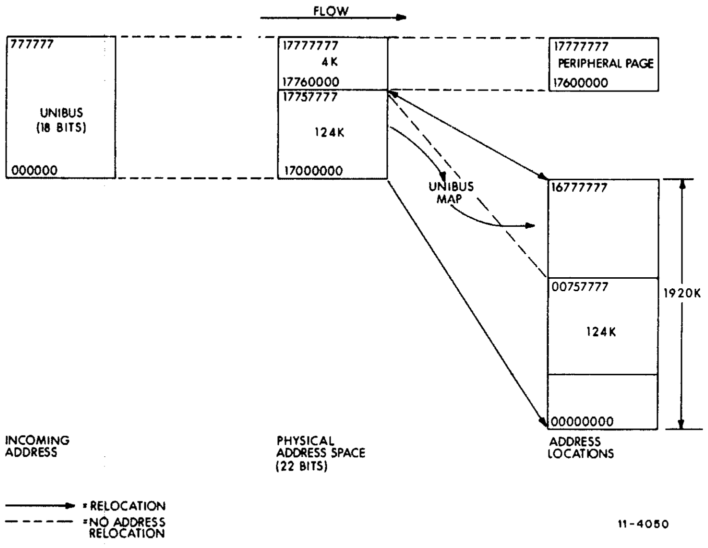


Figure I-6 Unibus Map Address Space

Software written for the PDP-11/20 or 11/45 runs without modification on the PDP-11/70 because the Unibus Map should not be enabled by this software, and because Memory Management will be enabled as required, i.e., the 18-bit mode enabled or disabled, and the 22-bit mode disabled. This does not take into account the difference in speed between these processors.

UNIBUS MAP ADDRESSING MODES

The Map is enabled when bit 5 of the Memory Management Register #3 (MMR3) is set. MMR3 responds to PA 17 772 516. Refer to Figure I-6.

1. The Unibus Map never responds to the Peripheral Page addresses (17 760 000 - 17 777 777).
2. When the Map is disabled, Unibus addresses 000 000 - 757 777 reference. Main Memory addresses 00 000 000 - 00 757 777, i.e., they are not modified.
3. When the Map is enabled, a Unibus address in the range of 000 000 - 757 777 is relocated by adding to it the contents of a Mapping Register.

It should be noted that the operations mentioned in 2 and 3 above are subject to fixed upper and lower address limits. These limits may be changed by adding or removing jumpers in the Unibus Map.

The Unibus Map is described in Section V of this manual.

CHAPTER 1
GENERAL DESCRIPTION

Memory Management receives all Virtual Addresses generated by the program, relocates them if necessary and then transmits the physical addresses to the Cache or to the Unibus. Address modification is the main function of Memory Management. This modification of addresses is called Relocation because it consists of adding a fixed constant to every virtual address (Refer to Chapters 2 through 4).

Memory Management also allows the user to protect one section of memory from access by programs located in another section. It divides the memory into sections - called pages (Chapter 8). Each individual page has a protection or access key associated with it that defines access to the page. With the Memory Management unit, a page can be keyed non-resident (memory neither readable nor writable) or read-only (no write operations to memory). These two types of protection, in association with other features, enable the user to develop a secure computer operating system. With the non-resident key, memory not specifically assigned to a program can be made unavailable to it (Chapter 9).

It is often desirable to load a program into one area of physical memory and then execute it as if it were located in another area of memory, e.g., when several user programs are simultaneously stored in memory. When any one program is running, it must be accessed by the processor as if it were located in the set of addresses beginning at 0. This process is called Relocation. When the processor accesses virtual address 0, a base address is added to the address; thus, the relocated 0 location of the program is accessed. Typically, this same base address is added to all references while the program is running. A different base address is used for each of the other programs in memory.

Memory Management specifies relocation on a page basis, which allows a large program to be loaded into discontiguous pages in memory. This ability eliminates the need to shuffle programs to accommodate a new one. It also minimizes unusable memory fragments, allowing more users to be loaded in a specific memory size.

A program and its data may occupy as many as 16 pages in the memory. The size of each page may vary and can be any multiple of 32 words, up to 4096 words in length. This feature allows small areas in memory to be protected, i.e., stacks, buffers, etc., and also allows the last page of a program, exceeding 4K words, to be of adequate length to protect and relocate the remainder of the program (Chapter 8). As a result, the memory fragmentation problem inherent with fixed-length pages is eliminated. The base address of each page can be any multiple of 32 words in the Physical Address space, thus ensuring compacted core. Finally, the variable page size enables pages to be dynamically changed at run time.

The Memory Management unit provides two bits of active page status information: an "accessed" bit and a "written into" bit. These bits can be used by the operating system program to determine whether the page has been accessed and, if so, whether it was written into. The accessed bit can be used by operating system programs to determine which page should be overlaid with the new program page in systems that swap programs back and forth from a disk. The written into bit can be used to determine whether the page to be overlaid must be swapped back to the disk or whether it is identicalto a copy already there.

Memory Management provides three separate sets of pages for use in the processor's Kernel, Supervisor, and User modes. These sets of pages increase system protection by physically isolating User programs from service Supervisor programs and the Kernel program. The service programs (compilers, editors, file system, assemblers, etc.) are also separated from the Kernel program (exception handling, I/O, memory management, etc.). Separate relocation register sets greatly reduce the time necessary to switch context between mapping. The three sets also aid the user in designing an operating system that has clearly defined communications, is modular, and is more easily debugged and maintained. During development cycles, these features result in time and cost savings; in the final system design, they result in an efficient and reliable system.

The Virtual Address space is further divided, within each of the Kernel, Supervisor and User pages, into Instruction Space and Data Space (I and D space). I space contains code, i.e., any word that is part of the program, such as instructions, index words and immediate operands. D space contains information that can be modified, such as data buffers.

By using this feature, Memory Management can relocate data and instruction references with separate base address values; thus, it is possible to have a user program of 64K words consisting of 32K of instructions and 32K of data. Moreover, a convenient means of building reentrant shared programs is provided (these programs keep a separate data area for each user). The ability to relocate data with separate base address values enables shared compilers, assemblers, editors, and supervisors to be developed.

PDP-11 stacks expand by pushing words into lower addresses and thus growing downward; procedure sections increase by growing into higher addresses. All memory pages can be expanded downward or upward by adding lower addresses (stack) or higher address (procedure, data). As a result, it is easy to expand both stack and program pages.

An Abort is the non-completion or interruption of a data cycle due to an error. Abcrts are serviced immediately, prior to the completion of the instruction during which they occur. A Trap is an interruption of the normal program flow by internal machine conditions. These conditions can be, but are not necessarily, errors. A trap is executed after the instruction during which it occurs is completed. Both aborts and traps generated by Memory Management transfer control to location 250. Three status registers (Chapter 9) record all information necessary to recover from a Memory Management abort. This information includes the page number that faulted, the type of violation that caused the fault (exceeded length, read-only violation, etc.), and all information needed to easily restart the aborted instruction once the Virtual Address has been corrected.

Three protection keys cause a trap, i.e., an automatic transfer of program control to location 250 at the end of the current instruction. The trap feature is useful for gathering "frequency of page use" statistics.

DEFINITION OF PAGE

A "Page" is a collection of contiguous addresses. Memory Management divides the 32K Virtual Address space into eight 4K sections called Virtual Pages. The lowest Virtual Address in each page is a whole multiple of 4096. The three high order bits of the VA [VA(15:13)] are the page number (0-7) and select a PAR/PDR pair within the current mode (Kernel, Super or User) and space (I or D).

This PAR/PDR pair in turn defines the Physical Page. The PAR contains the base address of this page, which may be on any whole multiple of 32 words. A block consists of 32 words, and a physical memory page may consist of up to 128 blocks. The Page Length Field (PLF) of the PDR (bits 14:08) determines the allowable length of the page. A page may expand upward (from lower to higher addresses) or downward. Expansion direction is determined by bit 3 of the PDR (ED).

PHYSICAL MEMORY PAGE

Refer to Figure 1-1. A block consists of $64_{10} = 100_8$ bytes or $32_{10} = 40_8$ words. The 6-bit word number (bits 05:00) field of the VA specifies an address (00 - 77) within the block (refer to Chapter 4).

A page consists of a maximum of 200_8 blocks (000 - 177), or $200_8 \times 100_8$ bytes = 20,000 bytes or 10,000 words. Thus, a page starting at PA = 00 000 000 has a maximum possible PA of 00 17 777. A block starting at 00 000 000 ends at 00 000 077.

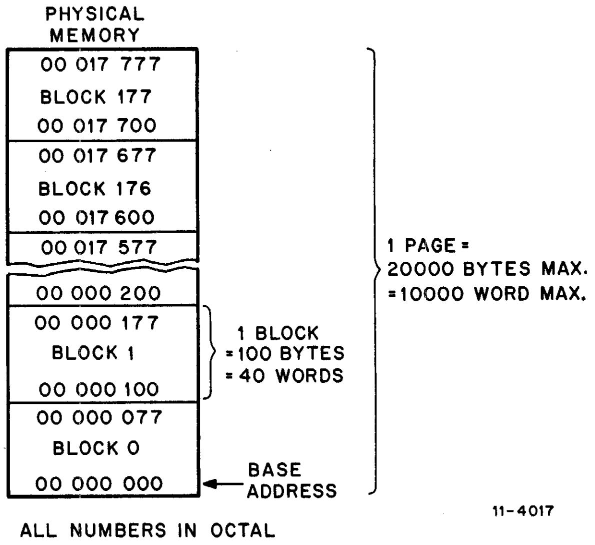


Figure 1-1 Example of Physical Memory Page

A Physical Address is constructed as follows (refer to Figure 1-2): the base address of the page is contained in the selected PAR. The block number field of the VA (bits 12:06) is added to this base address to give the base address of the block. The word number field of the VA specifies the Displacement In the Block (DIB).

The relocation example shown in Figure 1-3 illustrates several points about memory relocation:

1. Although the PAs appear to the program to be in contiguous address space, the 32K-word VA space is actually relocated to several separate areas of physical memory. As long as the total available physical memory space is adequate, a program can be loaded. The physical memory space need not be contiguous.

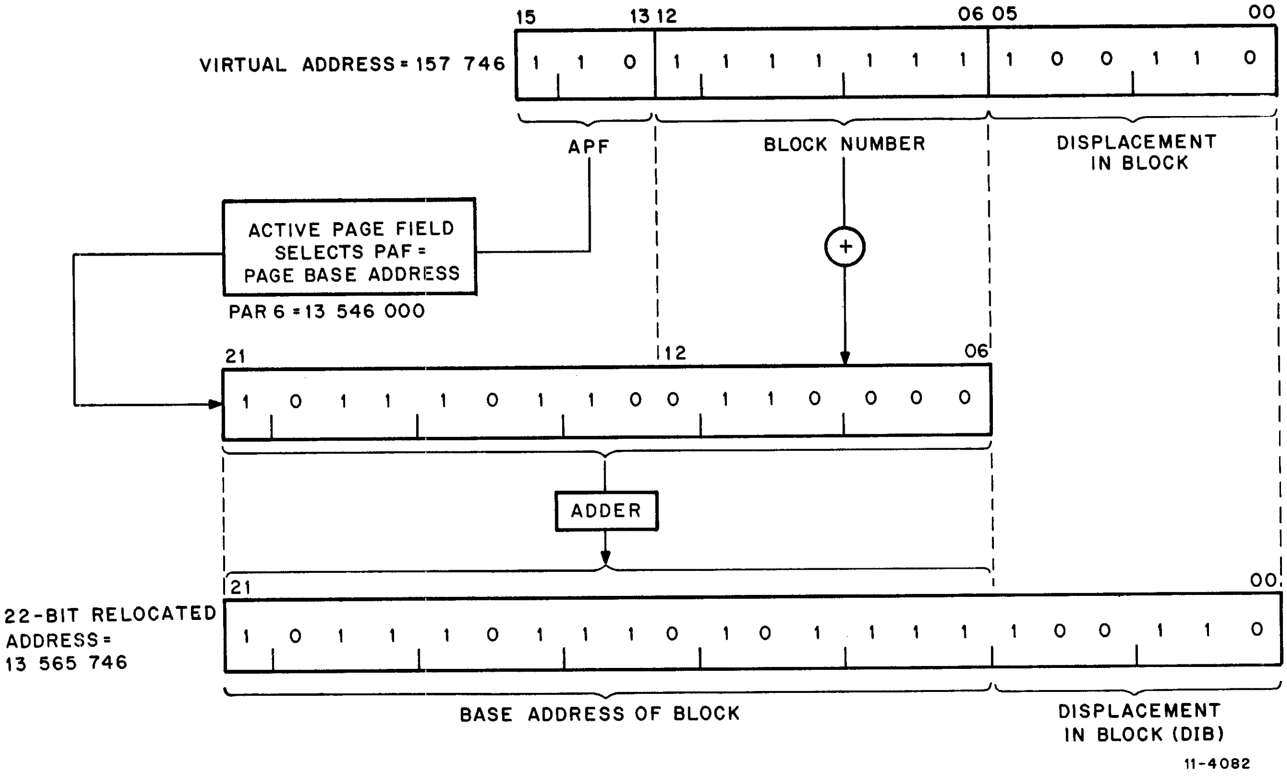


Figure 1-2 Construction of PA

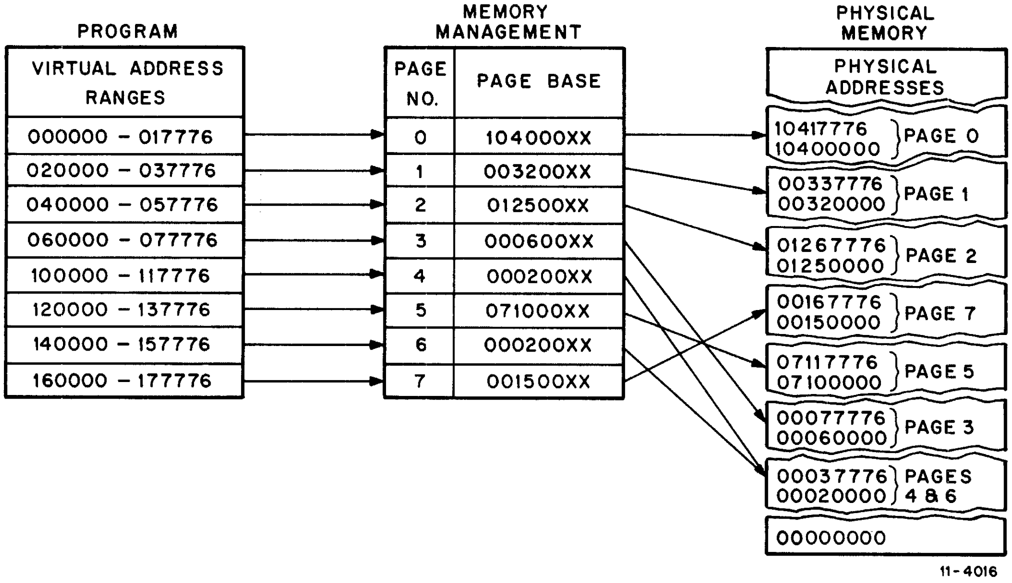


Figure 1-3 Relocation

2. Pages may be relocated to higher or lower PAs, with respect to their VA ranges. In the example, Page 1 is relocated to a higher range of PAs, Page 4 is relocated to a lower range, and Page 3 is not relocated at all, since the Page Base (=PAF) restores to the VA the three bits (15:13) which are stripped during relocation.

3. Each page is relocated independently. Two or more pages can be relocated to the same physical memory space. Using more than one page address register in the set to access the same space is one way of providing different memory access rights to the same data, depending on which part of a program was referencing that data. In Figure 1-3, note that the same relocation constant is assigned to Pages 4 and 6. As a result, VAs within both address ranges access the same PAs in memory, using different page address registers.

BLOCK DIAGRAM

Refer to Figure 1-4. Memory Management receives the VA from the processor. It generates the PA, which is received by the Cache or by the Unibus. As a result of its management functions, Memory Management informs the processor of traps and aborts.

Chapters 2 through 5 of this section describe the generation of the Physical Address. Chapters 6 through 9 explain the address checking and error reporting functions of Memory Management.

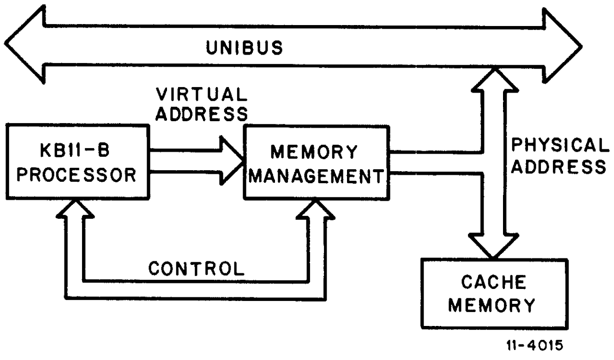


Figure 1-4 Block Diagram

CHAPTER 2
MEMORY MANAGEMENT MAPPING FUNCTION

When Memory Management is enabled, the normal 16-bit, direct-byte address is no longer interpreted as a direct Physical Address (PA) but as a Virtual Address (VA) containing information to be used in constructing a new 22-bit PA. The information contained in the VA is combined with relocation information contained in the Page Address Register (PAR) to yield a 22-bit PA. Using the Memory Management Unit, memory can be dynamically allocated in pages each composed of from 1 to 128 integral blocks of 32 words.

The starting PA for each page is an integer multiple of 32 words, and each page has a maximum size of 4096 words. Pages may be located anywhere within the PA space. The determination of which set of 16 page registers is used to form a PA is made by the current mode of operation of the CPU, i.e., Kernel, Supervisor or User mode.

2.1 CONSTRUCTION OF A PHYSICAL ADDRESS

All addresses with memory relocation enabled reference information in either Instruction (I) space or Data (D) space. I space is used for all instruction fetches, index words, absolute addresses and immediate operands. D space is used for all other references. I space and D space each have 8 PARs in each mode of CPU operation, Kernel, Supervisor, and User. Using Memory Management Register #3, the operating system may select to disable D space and map all references (Instructions and Data) through I space, or to use both I and D space.

The basic information needed for the construction of a PA comes from the VA, which is illustrated in Figure 2-1, and the appropriate PAR set.

The Virtual Address consists of:

1. The Active Page Field (APF). This 3-bit field determines which of eight Page Address Registers (PAR0-PAR7) will be used to form the PA.

2. The Displacement Field (DF). This 13-bit field contains an address relative to the beginning of a page. This permits page lengths up to 4K words (2^{13} = 8K bytes). The DF is further subdivided into two fields as shown in Figure 2-2.

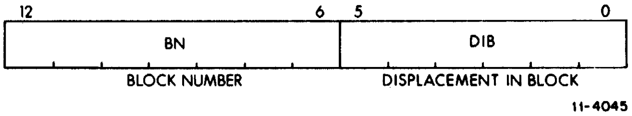


Figure 2-2 Displacement Field

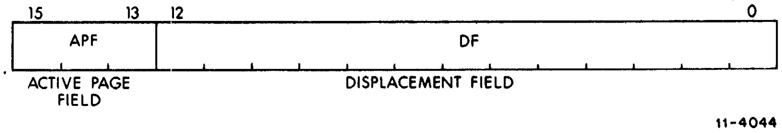


Figure 2-1 Interpretation of VA

The Displacement Field (DF) consists of:

1. The Block Number (BN). This 7-bit field is interpreted as the block number within the current page.
2. The Displacement in Block (DIB). This 6-bit field contains the displacement within the block referred to by the Block Number (BN).

The remainder of the information needed to construct the PA comes from the 16-bit Page Address Field (PAF), the Page Address Register (PAR) that specifies the starting address of the memory page which that PAR describes. The PAF is actually a block number in the physical memory, e.g. PAF=3 indicates a starting address of 96 (3 X 32) words in physical memory.

The formation of the PA is illustrated in Figure 2-3. The logical sequence involved in constructing a PA is as follows:

1. Select a set of PARs, depending on the space being referenced.
2. The APF of the VA is used to select a PAR (PAR0-PAR7).
3. The PAF of the selected PAR contains the starting address of the currently active page as a block number in physical memory.
4. The Block Number (BN) from the VA is added to the PAF to yield the number of the physical block in memory which will contain the PA being constructed.
5. The Displacement in Block (DIB) from the Displacement Field (DF) of the VA is joined to the physical block number to yield a 22-bit PA.

2.2 MANAGEMENT REGISTERS

Memory Management implements three sets of 32 16-bit registers. One set of registers is used in Kernel mode, another in Supervisor, and the other in User mode. The choice of which set is to be used is determined by the current CPU mode contained in the Processor Status word. Each set is subdivided into two groups of 16 registers. One group is used for references to Instruction (I) space, and one to Data (D) space. The I space group is used for all instruction fetches, index words, absolute addresses and immediate operands. The D space group is used for all other references, providing it has not been disabled by Memory Management Register

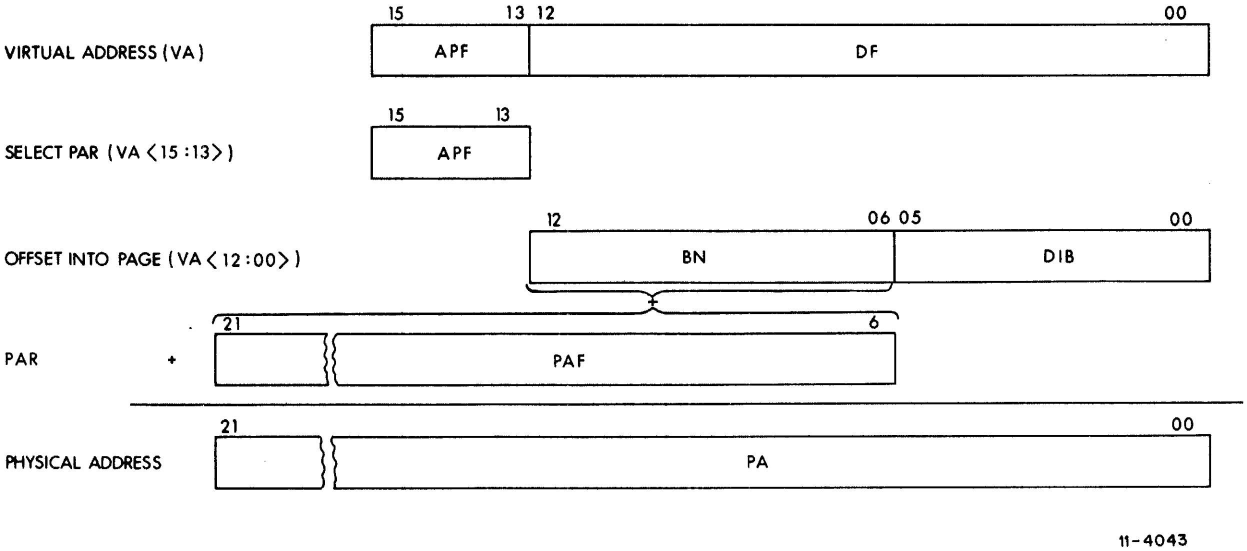


Figure 2-3 Construction of PA

#3. Each group is further subdivided into two parts of eight registers. One part is the PAR, whose function has been described in previous paragraphs. The other part is the Page Descriptor Register (PDR). PARs and PDRs are always selected in pairs by the top three bits of the VA. A PAR/PDR pair contain all the information needed to describe and locate a currently-active memory page.

The various Memory Management Registers are located in the uppermost 4K of PDP-11 PA space along with the Unibus I/O device registers.

This chapter and Chapters 3 through 6 describe the address relocation function of Memory Management, and the reading and writing of relocation registers by the program.

Refer to Figure 2-4. Relocation is essentially the process of adding the contents (PAF) of a register (PAR) to the program or VA. This sum is then modified, depending on the mapping selected, and becomes the PA.

Chapter 3 describes the selection of the PDR and PAR. The VA, the Processor Status Word (PSW or PS), the processor ROM address, and Memory Management Register #3 (MMR3) bits 2 - 0 (enable data space for User, Supervisor, or Kernel modes) make this selection.

Chapter 4 describes the generation of the PA. The PAF and the VA are summed (except in 16-bit mode), examined by the PA generation circuits, and output to the Unibus or to Memory.

Chapter 5 describes the generation of two functions: SAPN NOT CACHE ADRS and SAPN UNIBUS ADRS, which are used by other parts of the **KB11-C** for purposes of address checking for non-existent memory (NEXM) or control of the Timing Generator during bus cycles.

The contents of the PARs and PDRs are controlled by the program, which can load or read them.

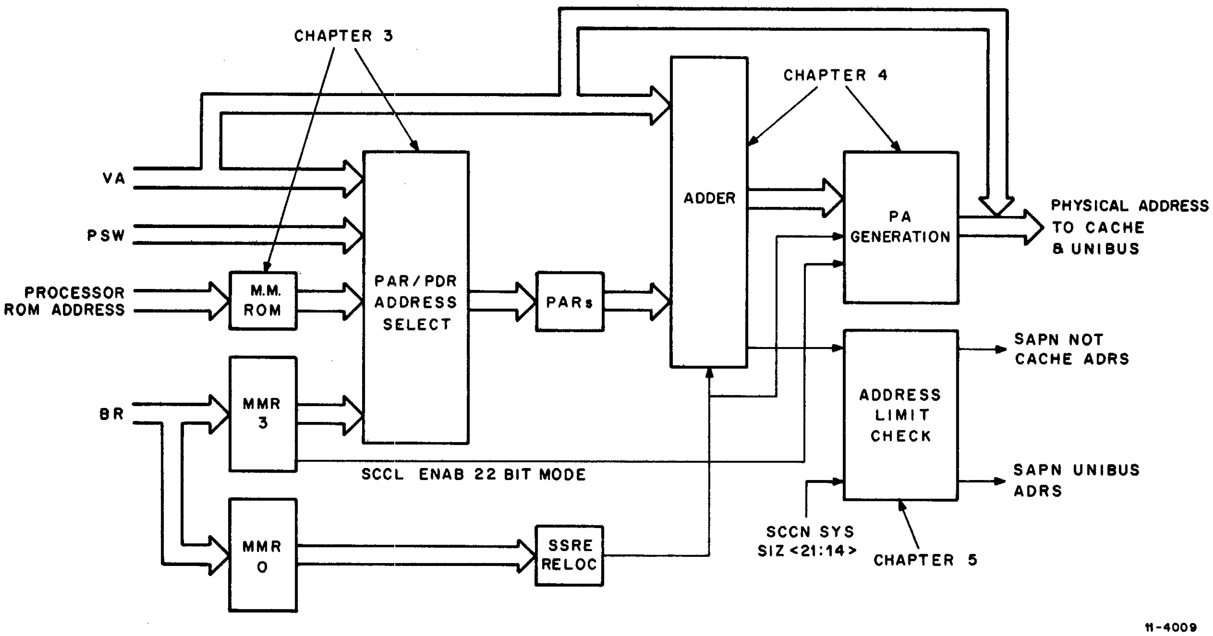


Figure 2-4 MM Relocation Function

Refer to Figure 2-5. Chapter 7 describes these read/write operations. The address decoders on the SCC module, which are also explained in this chapter, decode the incoming PA and select the PAR or PDR indicated by this address. The outputs of PARs and PDRs are driven onto the Internal Data Bus.

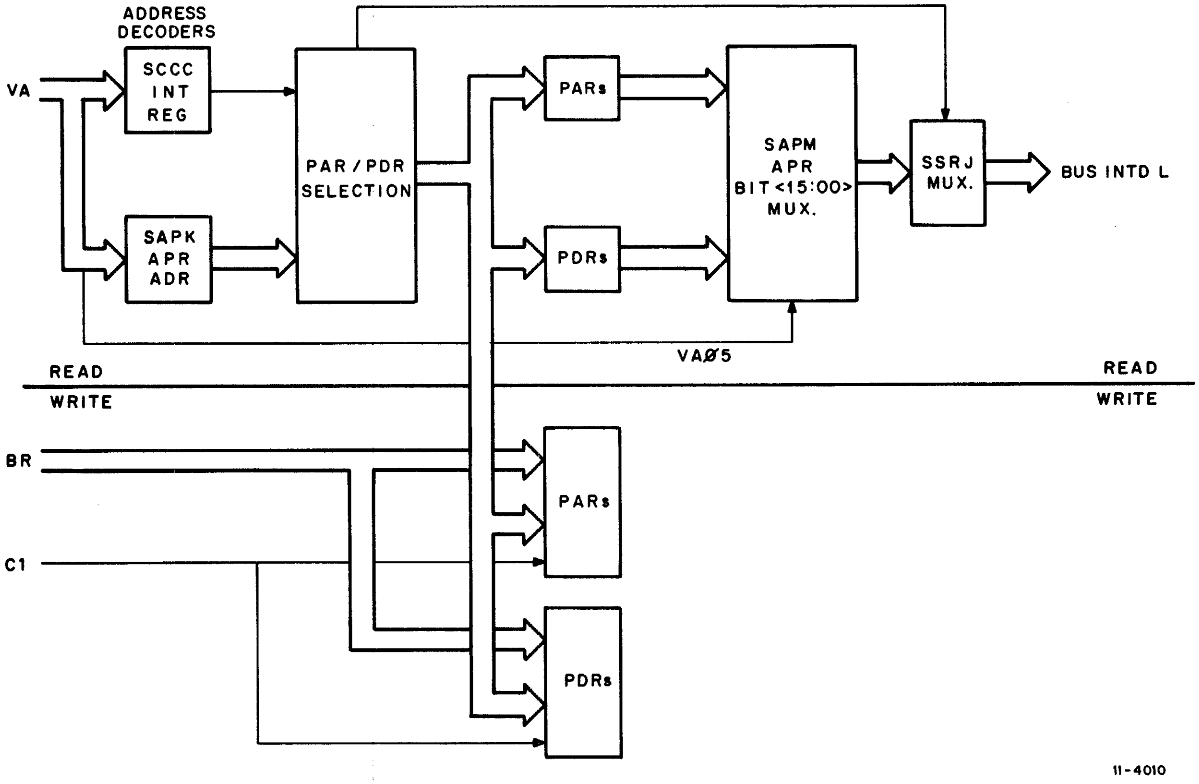


Figure 2-5 PAR/PDR Read/Write

CHAPTER 3
PAR AND PDR ADDRESSING DURING RELOCATION

NOTE
A working knowledge of the processor microprogram ROM (Section II, Chapter 1) is required for the understanding of this chapter.

Refer to Figure 3-1. There are 48 PARs and 48 PDRs, which are arranged in PAR-PDR pairs. The outputs of all PARs are wired-ORed [SAPA+B+C PAF(21:06)], as are those of the PDRs. Both the PAR and the PDR, in a pair, are selected and read at the same time.

Each mode (Kernel, Supervisor and User) has 16 PAR/PDR pairs available, eight for I space and eight for D space.

One of the Kernel, Supervisor or User PAR/PDR sets is selected by the PSW current mode bits [PS(15:14)] in conjunction with its previous mode bits [PS(13:12)] and the K, S, U space logic on drawing SSRB.

Either the I space set or the D space set for the current mode is selected by Memory Management Register #3, bits 2 - 0 (Enable K, S or U D space) in conjunction with the I space enable logic, which is also shown on drawing SSRB.

One of the eight PAR/PDR pairs in the selected set is then chosen by bits 15 - 13 of the Virtual Address.

3.1 MEMORY MANAGEMENT ROM (SSRA)

The Memory Management ROM shown on drawing SSRA controls many Memory Management functions. This ROM uses the same address [RACD RAR(07:00) H] as the processor microprogram ROM, and thus reflects the current state of the processor.

Drawing SSRL is a truth table of the output of the Memory Management ROM. The column headed Octal Location refers to the address of the processor ROM state listed in the second column. Truth Value ROMOUT columns 1 - 16 show, for each processor ROM state, the bits that are asserted by the Memory Management ROM. These bits are called SSRA ROM OUT(16:01) H. Refer to Section II, Chapter 1 of this manual for a detailed explanation of the processor ROM.

The two 74S157 multiplexers on SSRA are used as decoders for ROM OUT(03:01). Their output is clocked into the 74S174 flip-flops by SSRK PULSE23 H, which is a buffered TIGE TS2 L.

The functions generated by Memory Management ROM outputs 1 - 3 are discussed as required where they are used. ROM outputs 4 - 16 are used individually and are explained in the following paragraphs.

3.2 ROM OUTPUTS, 04 - 16

SSRA ROM OUT(04:16) inform the Memory Management logic of the occurrence of certain conditions in the processor. The significance of these outputs is described below.

ROM OUT04 (Destination Mode) is used during maintenance mode only; this bit is asserted during certain Destination Mode memory cycles. It is used in conjunction with bit 08 of MMR0 (Maintenance Mode) to enable relocation during the execution cycle of the instruction.

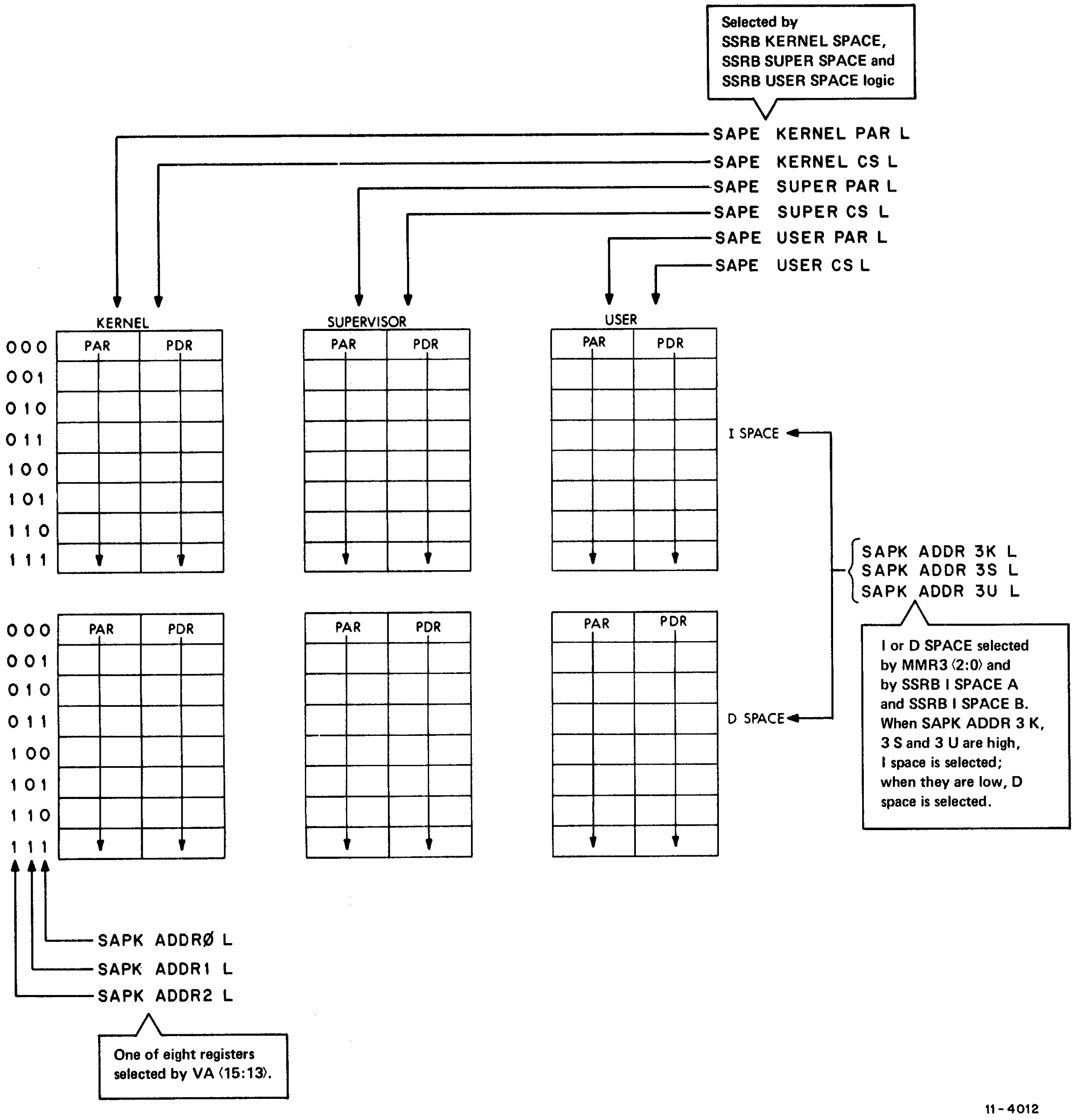


Figure 3-1 Addressing of PAR/PDR

ROM OUT05 [CLOCK PREV MODE (MT/FP)] is asserted during ROM cycles that assert BSOP1. This is shown on the Flows as BC-BSOP1. BSOP1 is normally called for in an execute cycle that is common to several instructions which have different bus operation requirements. Thus, BSOP1 is a DATI for MFP instructions and a DATO for MTP instructions. ROM OUT05 is used to clock the previous mode [PS(13:12)] into the K, S, U flip-flops during MFP and MTP instructions.

ROM OUT 06 (KERN DATI) is asserted when the BSC ROM bits=2 and forces Kernel mode during the Service Flows (Section II, Chapter 6). During these cycles, the new PC and PS are loaded from Kernel space. This condition appears on the Flows as BC-KERN DATI and occurs only in ROM cycles SVC.00 - SVC.30, BRK.30 and TRP.10. These cycles all force Kernel mode but do not change PS(15:14). BRK.30 and TRP.10 are followed by SVC.00, 10, 20 and 30. In this last cycle, the current mode bits, PS(15:14) are stored in the previous mode bits, PS(13:12), and the new current mode bits are loaded into PS(15:14) from BR(15:14) (IBS=2).

ROM OUT07 (BUST) is asserted during all BUST cycles. ANDed with T1, this bit is used as a clock in several places in Memory Management.

ROM OUT08 (I SPACE IF MT/FPI) is asserted during destination cycles that are used by MFP and MTP instructions, along with other instructions. I space is forced when this bit is asserted if the instruction is an MFPI or an MTPI [(IR15=0)*(IRCC MFP+MTP)]. See ROM OUT14.

ROM OUT09 (I SPACE ON IND WORD FETCH) is asserted during all fetch cycles and during cycles that read index words. Since these are all in I space, I space is unconditionally forced when this bit is asserted. ROM OUT10 is not used.

ROM OUT10 is not used.

ROM OUT11 (I IF INST START IN I) is asserted during the EXC.00, EXC.10, NEG.20 and SHR.10 microprogram cycles (Flows 11). These cycles execute the DATO/B portion of a DATIP/DATO transaction and thus must be executed in the same space as the previous DATIP data transfer cycle.

ROM OUT12 (DEPOSIT+EXAMINE) is asserted only during Console EXAM or DEP cycles (DEP.10, DEP.20, EXM.10, EXM.20, Flows 14). These cycles are executed in the space designated by the Console ADDRESS SELECT switch.

ROM OUT13 (SRCM=1+2+3+4+5) is asserted during cycles that fetch the source operand for binary instructions with source modes 1 - 5. If the source register field is 7, this operand is the second word of the instruction and as such is in I space. These cycles are S13.00, S13.01, S13.10 and S45.10 on Flows 1.

ROM OUT14 (DSTM=1+2) is asserted during Destination Mode 1 or 2 cycles that are common to MTP and MFP and to other DAC or O/class instructions. If the destination register field is 7, the operand is in I space. ROM OUT08 is also asserted when ROM OUT14 is asserted (D12.00, D12.01, D12.10, D12.60, D12.80 and D12.90, Flows 5), so that I space is addressed when ROM OUT14 is asserted

1. If the instruction is either MFPI or MTPI (I space is accessed, by definition), or
2. The Destination Mode of the instruction (not MFPI or MTPI) is 7 (the word accessed is in I space). See ROM OUT08.

ROM OUT15 (DSTM=3) is asserted during Destination Mode 3 cycles (D30.10, D30.80 and D30.90). If the destination register field is 7 during these cycles, the addressing mode is Absolute and the address word should come from I space.

ROM OUT16 (FLOATING POINT INST) is asserted for FPP Immediate Mode bus operations (FSV.00 and FSV.10, Flows 12). It is used to ensure that the immediate operand comes from I space if DM2 and DF7.

3.3 K, S, OR U MODE SELECTION (SSRB)

The chip select signals for PARs are SAPE KERNEL (or SUPER/USER) PAR L; for PDRs, these signals are SAPE KERNEL (or SUPER/USER) CS L. Because SCCC INT REG H is low during relocation and the B inputs to the multiplexers are selected, both the PAR and PDR chip select signals have the same source: SSRB KERNEL (or SUPER/USER) SPACE (1) L. These signals are the outputs of three flip-flops on SSRB that are clocked on the trailing edge of T1 of all BUST cycles (SSRB CLK SPACE H). The input to these flip-flops are the AND-OR-invert gates SSRB KS (or SS/US) L.

1. During a Console cycle (ROM OUT12), the Console ADDRESS SELECT switch determines the mode [SSRK CNSL KERNEL (or SUPER/USER) H].

2. A KERNEL DATI (ROM OUT06) unconditionally forces Kernel Mode during the Service Flows.

3. During the BSOP1 (execute) cycle (ROM OUT05) of an MFP or MTP instruction, the mode is forced to the previous mode, as determined by PS(13:12). Note that MFP instructions are I/class*DAC*BSOP1, which causes their destination cycle to be a DATI, and that MTP instructions are O/class*BSOP1, which causes their output cycle to be a DATO. IRCC MFP+MTP H is asserted during the execution of all MFPI, MFPD, MTPI and MTPD instructions.

4. K, S, or D space is selected by the current mode PS bits [PS(15:14)], if the current cycle is not the BSOP1 cycle (ROM OUT05) of an MFP or MTP instruction (SSRB MF/TP SPACE L), and if the current cycle is not a Console cycle (ROM OUT12). An additional condition that applies only to Super or User modes is that the current cycle not be a Kernel DATI (ROM OUT06).

3.4 I OR D SPACE SELECTION [SAPK ADDR 3 (K, S OR U) L]

SAPK APR ADDR3 K L, APR ADDR3 S L, and APR ADDR3 U L determine whether the I space or the D space PAR/PDR set is selected; when they are high, I space is addressed, when they are low, D space is addressed. The state of these bits is determined by bits 0, 1, 2 of MMR3 and by the I space logic on SSRB. During address relocation, SCCC INT REG A L is high, and the B inputs to the SAPK APR ADDR3 multiplexer are selected. An input to this multiplexer is high (thus selecting D space) if its corresponding MMR3 bit [SCCL INBL D K (or S/U) (1) H] is high and if I space is not required by the logic on SSRB. MMR3 is controlled by the program [BR(02:00)]. The output of the I space logic (SSRB I SPACEA L and SSRB I SPACEB L) is clocked on the trailing edge of T1 of BUST cycles (SSRA ROM OUT07 H) into the SAPK I SPACE flip-flops, which in turn are gated with the MMR3 outputs to generate the SAPK APR ADDR bits.

I space is forced whenever the output of either of the SSRB I SPACE gates is asserted (=low).

SSRB I SPACEA L is asserted under the following conditions:

1. During a Console DEP or EXAM (ROM OUT12) if the ADDRESS SELECT switch is in any of the I space positions (Kernel, Super or User) as indicated by SSRK CNSL I SPACE H.

2. During all instruction and index word fetch cycles (ROM OUT09).

3. During FPP Immediate Mode bus operations (ROM OUT16 and IRCC DSTM2 and SSRB DSTF7).

4. During Absolute Mode address word cycles (ROM OUT15 and SSRB DSTF7).

SSRB I SPACEB L is asserted under the following conditions:

1. During cycles that fetch the source operand for binary instructions in source modes 1 – 5 (ROM OUT13) when the source field is 7 (IRCB SRCF7). This includes Immediate and Absolute Modes.

2. During DATO or DATOB cycles that complete a DATIP operation (ROM OUT11), if the previous mode was I (SSRB PREV=I). This ensures that the DATIP-DATO/B operation is performed to the same memory location.

3. During destination cycles (ROM OUT08) when the instruction is either MFPI or MTPI [IRCC MFP+MTP and not IRCA IR15 (1) H], unless both current and previous modes are User and the instruction is an MFPI(IR07=0). In other words, the Destination Mode of MFPI and MTPI is executed in I space, except that the MFPI is executed in D space if both previous and current modes are User. This prevents a program from using MFPI to read from a read-only page in his I space, thus preserving the integrity of proprietary programs (Execute-Only=I space and read-only).

   For example – assume that a User program requests service from the Kernel program by doing an EMT. After this instruction, the mode bits in the PSW are:

   current [PS(15:14)] = Kernel
   previous [PS(13:12)] = User

   After executing the User program's request, the Kernel program returns control to the User program by an RTI. Before doing this, the Kernel program ensures that both current and previous mode bits are set to User. If this were not done, the User program could read the Kernel proprietary code via the MFPI.

4. During Destination Mode 1 or 2 cycles (ROM OUT14), and if the destination field is 7 (Immediate Mode) and the instruction is not MFP or MTP.

3.5 REGISTER SELECTION [SAPK ADDR(2:0) L]

The select bits for a PAR/PDR pair [SAPK APR ADDR(2:0) L] are common to all PARs and PDRs. SCCC INT REG A L is high because a PAR or a PDR is not directly referenced, and the B inputs to the multiplexers on SAPK are selected. SAPK APR ADDR(2:0) and APR ADDRA(2:0) are the same as BAMX(15:13) H. ADDRA(2:0), which are identical to ADDR(2:0), are used only for the 3101As that contain PAF(09:06), and are not buffered (as are the ADDR bits). The address is implemented in this manner to speed up the generation of bits 09 – 06 of the PA. These bits, together with VA(05:00) are the index field of the address input to the Cache.

CHAPTER 4
GENERATION OF THE PHYSICAL ADDRESS

In 16-bit mapping, the Virtual Address (VA) is not modified, and the relocated address is the same as the VA.

In 18- and 22-bit mapping, the VA is added to the contents of the selected PAR. This sum is the relocated address. [The contents of the selected PAR are also referred to as the Page Address Field (PAF)]. The logic that executes this operation is shown on drawing SAPJ.

In 16- and 18-bit mapping, the relocated address is examined to determine whether it is a Unibus address. If it is, the high order bits are set to 1s; if it is not, these bits are set to 0 to form the Physical Address (PA).

In 22-bit mapping, the PA is the same as the relocated address.

4.1 16-BIT MAPPING

Refer to Figure 4-1. In 16-bit mapping, the PA space consists of 28K memory locations (PA=00 000 000 - 00 157 777) and the 4K Peripheral Page (PA=17 760 000 - 17 777 777).

Physical Addresses 00 160 000 - 17 577 777 cannot be generated by the processor when using this mapping.

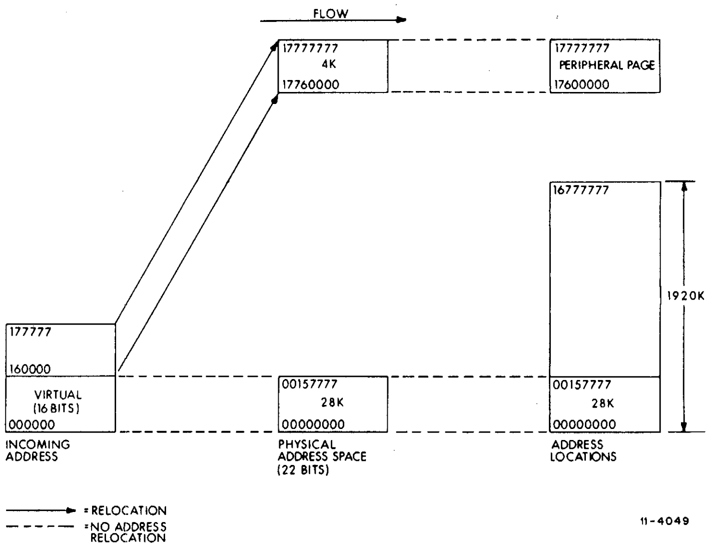


Figure 4-1 16-Bit Mapping

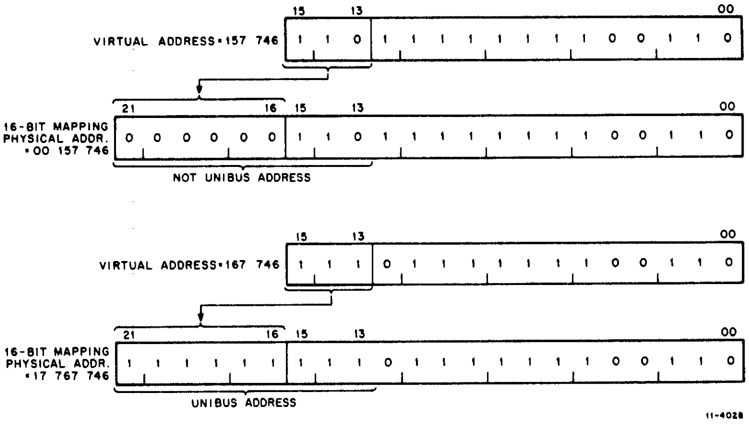


Figure 4-2 16-Bit Mapping: Generation of PA

Refer to Figure 4-2. A 16-bit VA is a PA if bits 15:13 are not equal to 111. In this case, bits 21:16 of the PA are made 0s, and bits 15:00 are the same as in the VA. If bits 15:13 of the VA are equal to 111, a Unibus address is intended by the program, bits 21:16 of the PA are made 1s, and bits 15:00 are unchanged from the VA.

4.2 VIRTUAL ADDRESS

The VA consists of three fields:

1. Bits 15:13, the Active Page Field (APF). These bits select one of PARs 0 - 7 within the mode and space selected.

2. Bits 12:06, the Block Number (BN). These bits are added to the PAF to form bits 21 - 06 of the PA.

3. Bits 05:00, the Displacement in Block (DIB). These bits are not altered and become bits 05 - 00 of the PA.

4.3 18-BIT MAPPING

Refer to Figure 4-3. In 18-bit mapping, the VA is added to the selected PAF to generate the PA. This address has a range of 128K, from address 00 000 000 - 17 777 777.

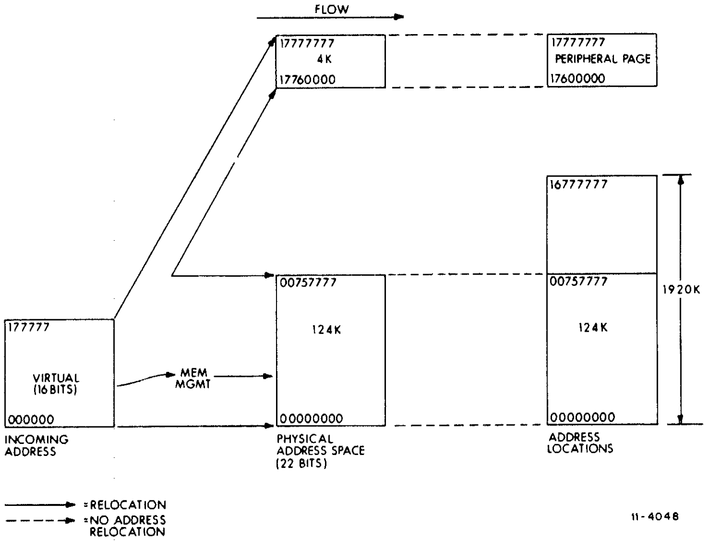


Figure 4-3 18-Bit Mapping

If bits 17:00 of the PA are 000 000 - 757 777, it is a Memory reference and PA(21:18) are forced to zeroes. If PA(17:00) are 760 000 - 777 777, it is a Unibus reference and PA(21:18) are forced to ones.

Physical Addresses 00 760 000 - 17 757 777 cannot be generated when using this mapping. Refer to Figures 4-4 and 4-5, which show examples of 18-bit PA generation.

Figure 4-4 shows the case of an 18-bit PA that is not a Unibus reference, i.e., bits 17:13 are not all 1s. In this case, bits 21:16 of the PA are modified to zeroes, which causes a memory reference.

Figure 4-5 shows the case of an 18-bit relocated address that is a Unibus reference, i.e., bits 17:13 are all 1s. In this case, bits 21:16 of the PA are changed to 1s, which causes a Unibus reference.

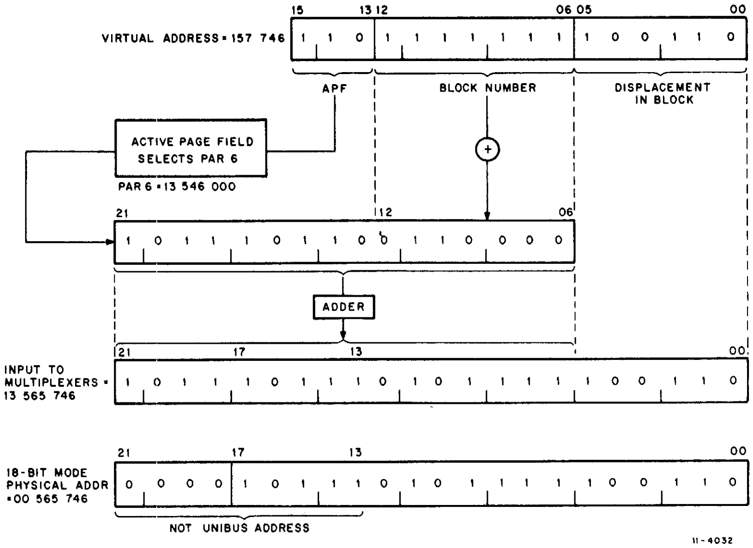


Figure 4-4 18-Bit Mapping: Cache Address

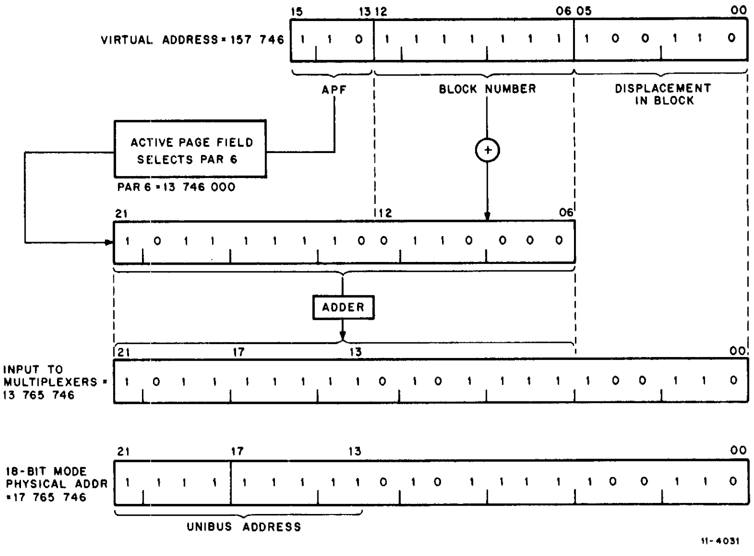


Figure 4-5 18-Bit Mapping: Unibus Address

4.4 22-BIT MAPPING

Refer to Figures 4-6 and 4-7. In 22-bit mapping, the VA is relocated in the same manner as in 18-bit mapping, but the relocated address becomes the PA without modification. Thus, all PAs from 00 000 000 - 17 777 777 can be generated.

Addresses 17 760 000 - 17 777 777 are Unibus I/O Page references. Addresses 00 000 000 - 16 777 777 are memory references. The 124K of addresses from 17 000 000 - 17 757 777 may be used to access memory via the Unibus Map (dotted lines on drawing).

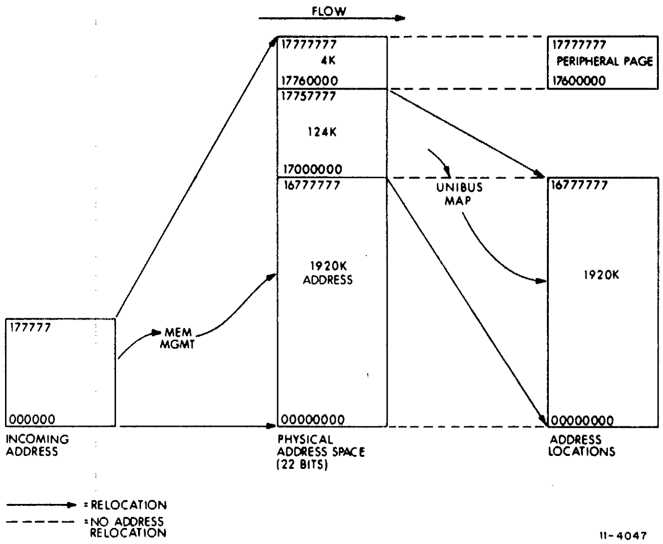


Figure 4-6 22-Bit Mapping

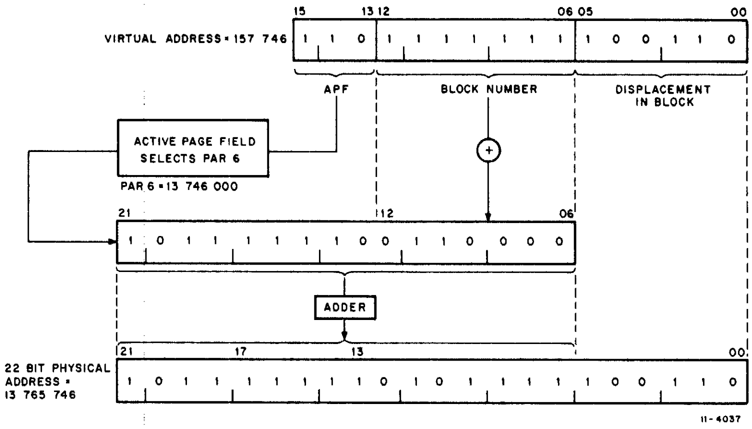


Figure 4-7 22-Bit Mapping

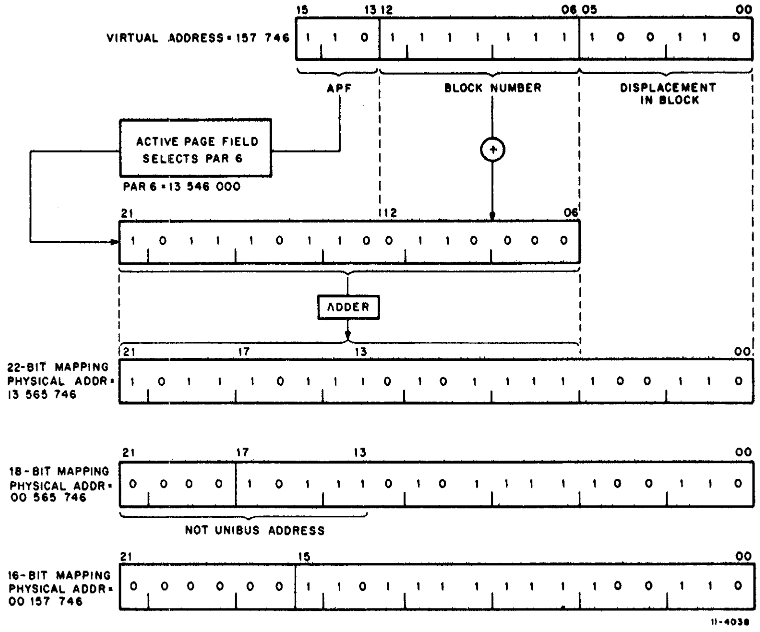


Figure 4-8 Physical Address Generation: Example 1

Refer to Figures 4-8 and 4-9. It should be noted that if the mapping is changed, the PA may also be changed. In Figure 4-8, three different PAs are generated from the same VA and PAF:

1. In 22-bit mapping: 13 565 746

2. In 18-bit mapping: 00 565 746

3. In 16-bit mapping: 00 157 746

These PAs are all memory references.

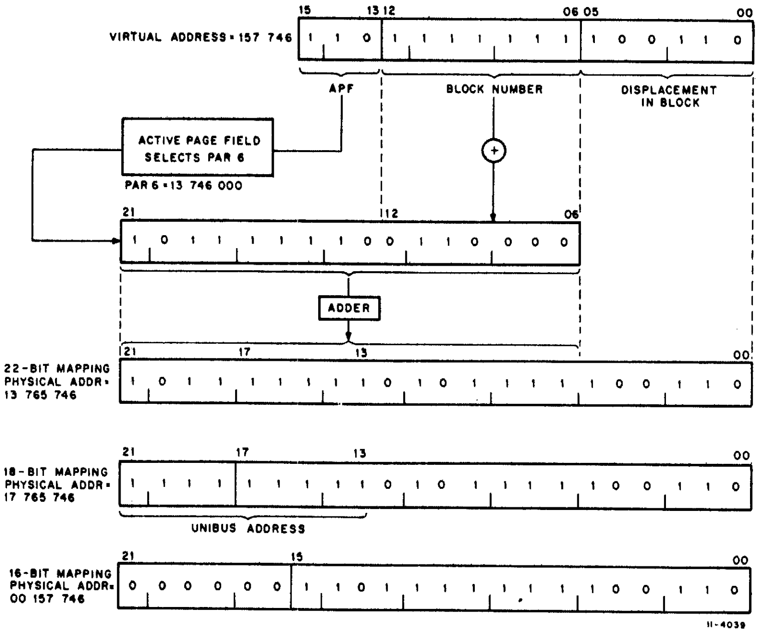


Figure 4-9 Physical Address Generation: Example 2

In Figure 4-9, also using the same VA, three different addresses are generated, two of which are memory references and one a Unibus reference:

1. In 22-bit mapping: 13 765 746

2. In 18-bit mapping: 17 765 746

3. In 16-bit mapping: 00 157 746

4.5 RELOCATION LOGIC

The relocation logic shown on schematic SAPJ is controlled by three functions:

1. SSRA KY PH MEM AC (1) H, which is a flip-flop that is set during all ROM Console EXAM or DEP cycles if the ADDRESS SWITCH is in PROG PHY or CONS PHY.

2. SCCL ENAB 22BIT MODE H, or bit 04 of MMR3 (address 17 772 516), which is controlled by the program.

3. SSRE RELOC L, which is controlled by bits 00 and 08 of MMR0 (address 17 777 572). These bits are also generated by the program. Bit 00 causes RELOC to be asserted when SSRA KY PH MEM AC (refer to 1 below) is cleared. Bit 08 allows one additional condition to assert RELOC: SSRA DST (1) H set; this flip-flop is set on the trailing edge of T1 of bus cycles that write into a destination address. (See 5 below.)

   RELOC controls the ALU function, as it is the S0 and S3 control input to the 74S181 ALU ICs. The three functions are combined to generate SAPJ SEL0 and SEL1 H, which control the four PA multiplexers.

Refer to Figure 4-10.

1. Console Mapping
   If SSRA KY PH MEM AC (1) H is set, both SEL1 and SEL0 are high and Memory Management is in Console Mapping. The D inputs to the 74S153 multiplexers and the B inputs to the 74S157 are selected. SAPJ PA(21:16) equal SCCK SWR(21:16), PA(15:13) equal VA(15:13) [SWR(15:00) are stored in the SR during a LOAD ADRS and read back via the BAMX]. Since RELOC is negated in Console mapping, PA(12:00) also equal VA(12:00).

2. 16-BIT Mapping
   If KY PH MEM AC is cleared and RELOC is not asserted, SEL1 is high and SEL0 is low. The C inputs to the 74S153s are selected. If SAPH EX MEM FLAG H is high, a Unibus address is required [VA(15:13)=111], and PA(21:16) become all 1s; if VA(15:13) are not all 1s, EX MEM FLAG is low, PA(21:16) become all 0s. In both cases, PA(15:00) equal VA(15:00), since the B inputs to the 74S157 are selected and the ALU function is A (RELOC not asserted).

3. 18-BIT Mapping
   If RELOC is now asserted and SCCL ENAB 22BIT MAPPING H is not (18-bit mode), the ALU mode becomes A+B and all addresses are relocated. SEL1 and SEL0 are both low, the B inputs to the 74S153s are selected, and the ALU function is A+B. If PA(17:13) are all 1s (Unibus address), PA(21:18) also become all 1s. If PA(17:13) are not all 1s, PA(21:18) become all 0s (memory reference). In both cases, the remainder of the PA equals the output of the ALU (bits 15:13 are selected through the 74S157 multiplexer).

4. 22-BIT Mapping
   If ENAB 22BIT MAPPING is now asserted, SEL1 is low and SEL0 becomes high, thus selecting the A inputs to the 74S153s. The A~~L~~U function is A+B. The output of the ALU becomes the PA without modification.

5. Destination or Maintenance Mapping
   If MMR0 bit 00 is cleared and bit 08 is set, Memory Management operates in 16-bit mapping, except during certain destination mode ROM cycles, which it executes in either 18-bit or 22-bit mapping (depending on the state of SCCL ENAB 22BIT MAPPING). The ROM cycles during which this occurs are those for which the Memory Management sub-ROM bit 04 is asserted.

This mapping should only be used for diagnostic purposes.

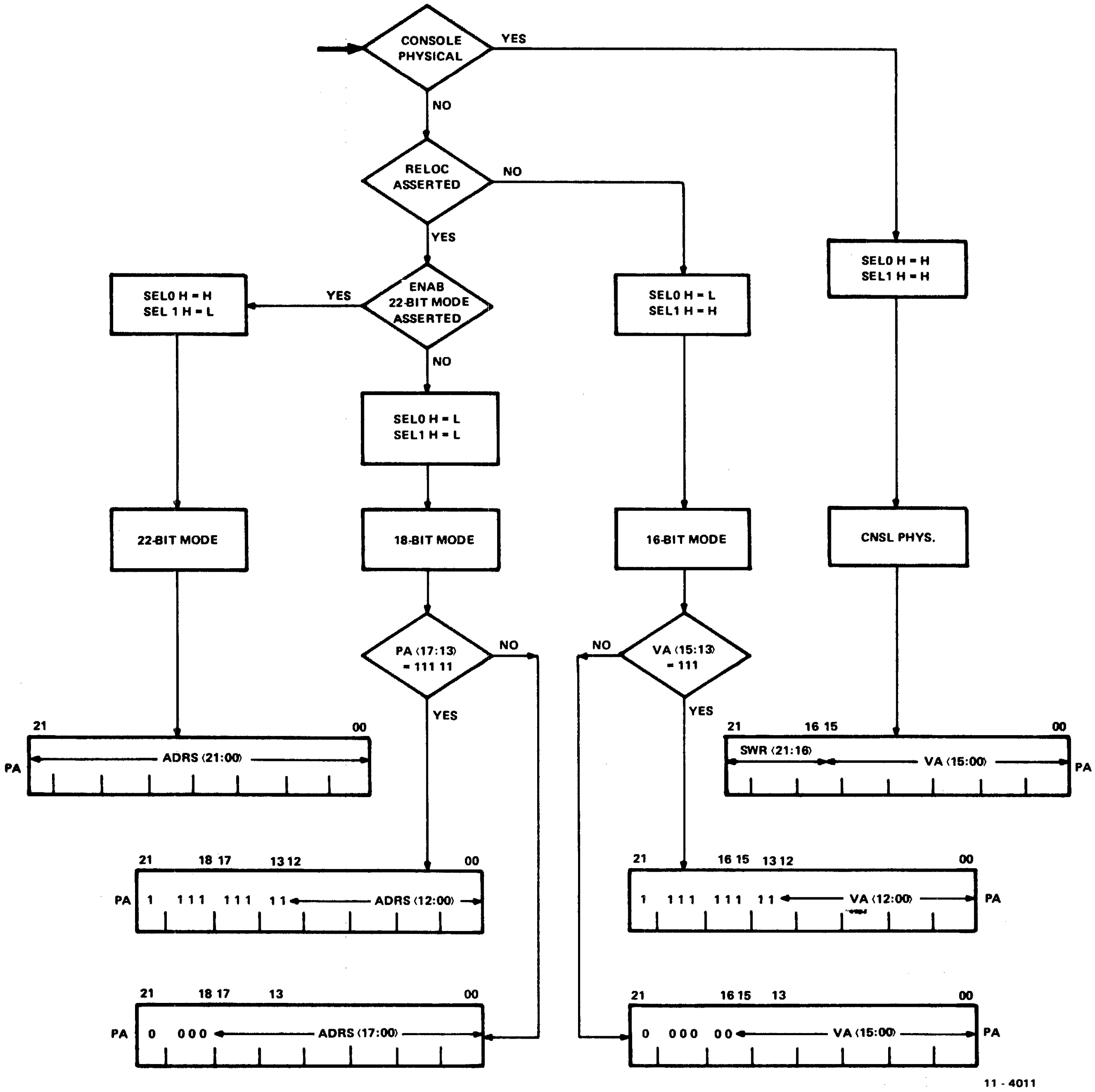


Figure 4-10 Generation of Physical Address

ROUTING OF PHYSICAL ADDRESS

1. Unibus Drivers – SAPJ PA(17:12), SCCA PA(11:06), and SCCA VA(05:00) are input to the Unibus drivers on SCCL. The gating function is UBCA CPBSY B H, which is asserted 150 ns before the processor MSYN. The output of the drivers is BUS A(17:00) L.

2. Cache – SAPJ PA(21:06) are an input to the Cache address multiplexer, ADME AMX(21:06). The Cache receives address bits 05 – 00 directly from the BAMX.

CHAPTER 5
ADDRESS VALIDITY

Memory Management examines an address for the purpose of determining whether it is a Unibus address, or a valid Cache address. The signals generated as a result of this examination are used by the TMC and UBC modules during data transfer operations

5.1 UNIBUS ADDRESS

SAPN UNIBUS ADRS L is asserted whenever the PA points to a Unibus reference.

The four AND inputs to SAPN UB ADRS L each decode the Unibus address for the four mapping modes:

1. In 18-bit mapping, when PA bits (17:13) are all 1s;

2. In 22-bit mapping, when PA(21:18) are all 1s;

3. In 16-bit mapping, when SAPH EX MEM FLAG H is high. [i.e., when VA(15:13) are all asserted]

4. In Console mapping, when switch register bits (21:18) are all 1s.

The output of SAPN UB ADRS L is ORed with SCCC INT REG B L, which decodes PAR and PDR addresses. These are Unibus addresses and as such are decoded by SAPN UB ADRS L; SCCC INT REG B L, however, is stable until T1 of the next BUST cycle, and keeps SAPN UNIBUS ADRS L asserted if the reference was to a Memory Management register.

5.2 NOT CACHE ADDRESS

SAPN NOT CACHE ADRS is used to notify the CP that the address generated by Memory Manage-'ment does not exist in the Cache. This signal is the result of a comparison between the PA and the Size Register.

1. 16-bit Mapping – All 16 bit mapping references are legal memory references or Unibus addresses; therefore no comparisons are necessary.

2. 18-bit Mapping – If there is more than 128K of memory on the system, then all addresses that can be generated are valid addresses. If, however, there is less than 128K of memory, invalid addresses are possible and must be tested for.

3. 22-bit Mapping – The PA is checked against the Size Register.

Two arithmetic signals are used to generate NOT CACHE ADRS: OVERFLOW and WRAPAROUND.

WRAPAROUND is asserted and disables NOT CACHE ADRS if there is a carry out of the MSB of the PA being generated.

e.g., 18-bit mapping:

If the PAR contains 007 777 and the VA is 000 100, the address generated is 00 000 000 (a valid memory reference) and 18 BIT WRAPAROUND L is true.

e.g., 22-bit mapping:

If the PAR contains 177 777 and the VA is 000 100, the address generated is 00 000 000 (a valid memory reference) and 22 BIT WRAPAROUND L is true.

OVERFLOW H is asserted when the PA is greater than the highest legal address in memory.

The first time OVERFLOW H is asserted is when the PA is greater than the value in the Size Register, i.e., the PAR is equal to the Size Register and a VA of 000 100 is used. For instance, if the Size Register contains 5777, there is 96K of memory. If the PAR contains 5777 and the VA equals 000 100, the PA generated is 00 600 000, which is the first nonexistent memory address.

SAPN NOT CACHE ADRS H is asserted when either SAPN UB ADRS L or the AND-OR-invert gates are asserted. This last gate is asserted if the PA is above the size boundary.

This is checked by the WRAPAROUND and OVERFLOW functions, except in the case of SCCC INT REG H.

WRAPAROUND is generated by the relocation logic on SAPJ for 18-bit and for 22-bit mapping. SAPJ 18-BIT WRAPAROUND L is the carry output of bit 17 of the adder; 22 BIT WRAPAROUND is the carry output of bit 21. Refer to Figure 5-1. A carry can only be generated at bit 21 when the PAF bits (21:13) are all 1s and a carry is generated by the sum of bits 12 of the PAF and the VA. Figure 5-1 shows the generation of the greatest PA possible with WRAPAROUND : 00 017 6(77). This is a Cache address. The 18-bit WRAPAROUND also generates the same maximum address, which is also a Cache address in 18-bit mapping.

OVERFLOW is generated on SAPN for 18- and 22-bit mapping and on SCCN for Console mode to determine if the address is greater than the System Size Boundary. In both cases, the adders are used as comparators and only the carry output is used. A 1's complement subtraction is implemented in both cases. A number Q is subtracted from another number P by adding the 1's complement of Q to P. A carry is generated only when Q<P, as illustrated by the following examples:

	Q<P		Q=P		Q>P	
P	5	101	5	101	5	101
Q	-3	+100	-5	+010	-6	+001
		1 001		0 111		0 110
	carry		no carry		no carry	

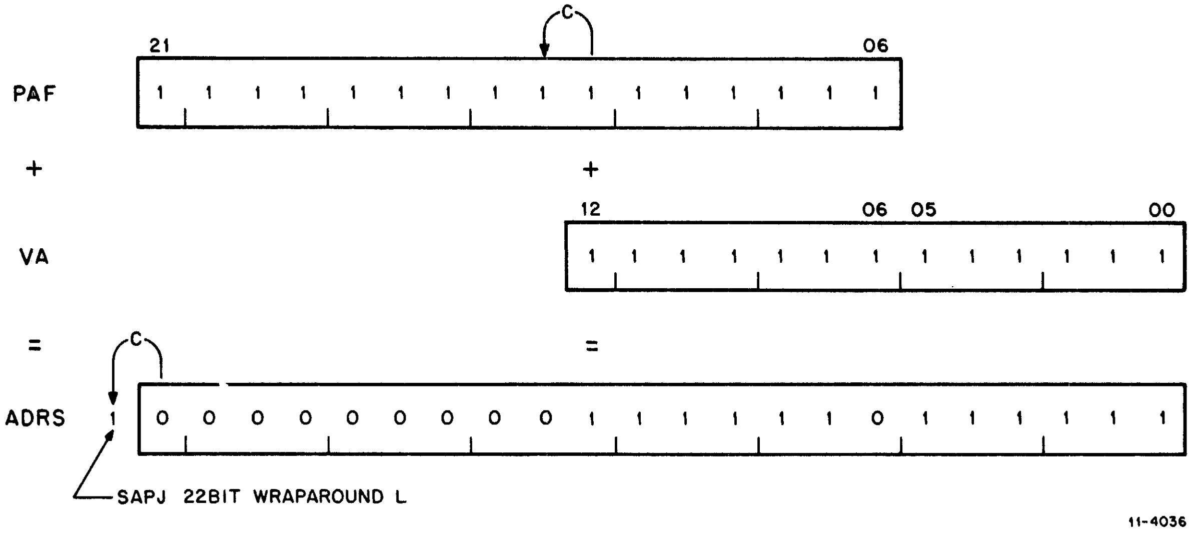


Figure 5-1 Wraparound

The carry therefore flags a Q<P condition which indicates a legal memory reference.

Overflow is generated for 18- and 22-bit mapping on SAPN. There is an overflow if the PA is greater than the system size: PA>SIZ. Refer to Figure 5-2. This is tested by the function

$$PA\text{-}SIZ$$

But

$$PA = (PAF+VA)$$

therefore

$$PA\text{-}SIZ=(PAF+VA)\text{-}SIZ \text{ or } PA+(VA\text{-}SIZ)$$

Since the subtraction is done by adding the complement of the subtrahend to the minuend,

$$PA\text{-}SIZ=PAF+[VA+(notSIZ)]$$

This is the function implemented by the adder on SAPN:

1. PAF(12:06) are added to VA(12:06). The ALU function is A plus B.
2. PAF13 is added to 0. The ALU function is A plus B.
3. SCCN SYS SIZ(21:14) is subtracted from PAF(21:14). The ALU function for these bits is A minus B, which is accomplished internally by adding A to the 1's complement of B.

For Console mapping, overflow is generated on SCCN. SCCN CONS OVERFLOW H is the inverted carry output of an 8-bit adder, the inputs to which are the System Size Register and the negation of the Console switch address. Bits (15:14) of the Switch Register are read from SCCA VA(15:14) C L, while bits (21:16) are read directly from the switches; this is because bits (21:16) of the Switch Register are loaded into the SR during a LOAD ADRS cycle, and read from the BAMX.

Refer to Figure 5-3. The arithmetic operation consists of summing the System Size Register with the negation of the switch address, and taking the negation of the final carry (=borrow) as the indication of an OVERFLOW. This operation gives the same result as would subtracting the System Size Register from the Switch Register and taking the non-inverted carry as the indication of an OVERFLOW.

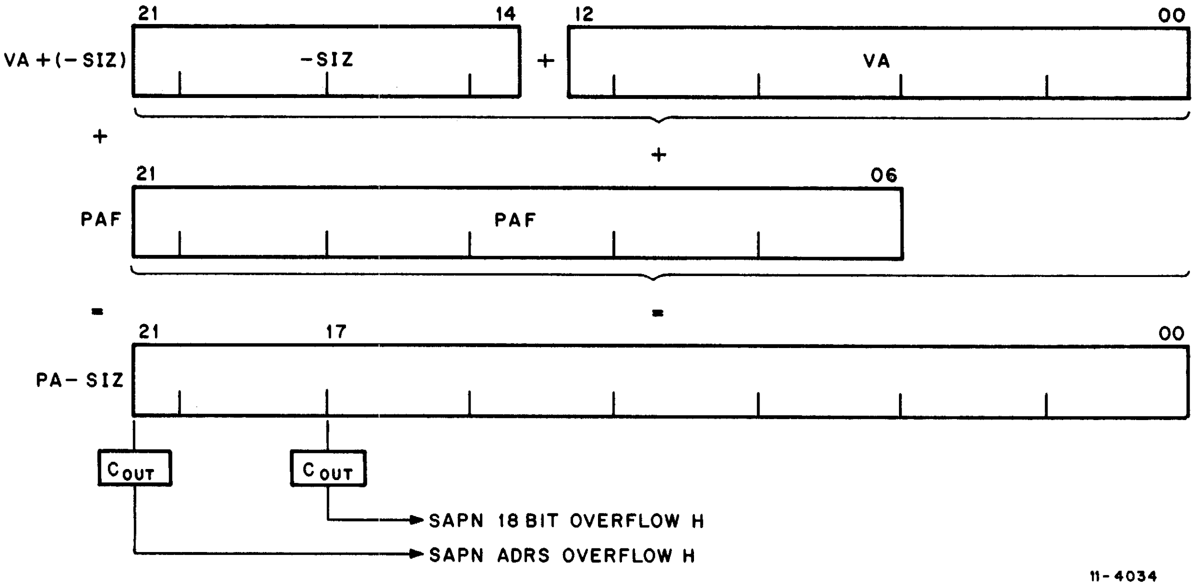


Figure 5-2 18- and 22-Bit Overflow

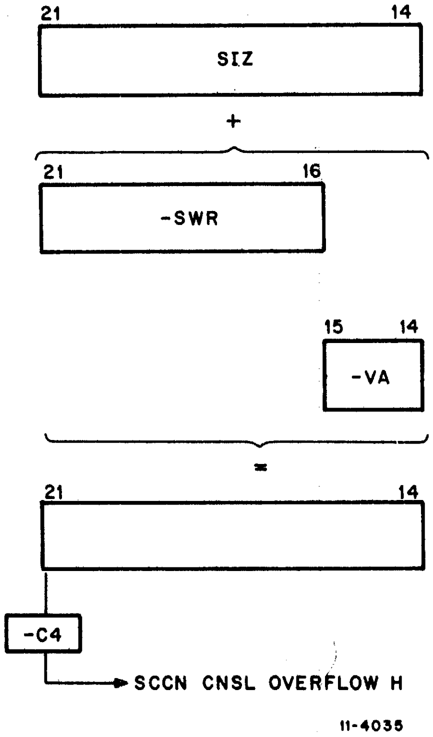


Figure 5-3 Console Overflow

5.2.1 18-Bit Mapping

The logic for 18-bit mapping, SAPN NOT CACHE ADRS, is the same as that for 22-bit mode with the exception of the added wired-OR gate; the output of this gate is high only when in 18-bit mode (SAPJ SEL0 and SEL1 H both low) and when the System Size Boundary is less than or equal to 00 777 777 [SYS SIZ(21:18) are all 0s]. If this is true, there is less than 128K of memory in the system. In this case, the 18-bit input AND gate to the AND-OR-invert gate is enabled, and an OVERFLOW with no WRAPAROUND means that the address is too high, and thus not a Cache address. The output of the gate is low either when not in 18-bit mapping, or if the System Size is greater than 00 777 777. Since this address is the greatest that can be generated in 18-bit mapping, OVERFLOW is meaningless and the 18-bit mapping gate is disabled.

5.2.2 22-Bit Mapping

If there is a 22-bit OVERFLOW (SAPN ADRS OVERFLOW H asserted), and if there is no WRAPAROUND (SAPJ 22-BIT WRAPAROUND L is high), then the address is not a Cache address in 22-bit mode.

5.2.3 Console Mapping

The PA is not a Cache address if, during a Console DEP or EXAM operation with the address switch in either of the PHYSICAL positions, the Switch Register contains an address greater than the System Size Boundary (SCCN CNSL OVERFLOW H).

SSRA KY PH MEM ACC is the output of a flip-flop, clocked at every T1, whose input is the AND of SSRA ROM OUT 12 and SSRK CNSL PHY ADRS H. The first function is asserted only during EXM or DEP ROM cycles; the second when the ADDRESS SELECT switch is in either PROG PHY or CONS PHY positions. Note that PROG PHY is used only for readout and is meaningless during a DEP (write) Console operation. Refer to Section III, Chapter 1.

CHAPTER 6
DESCRIPTION OF PDR

In addition to its relocation function, Memory Management has supervisory or memory protection functions.

The Page Description Register (PDR) is read at the same time as its corresponding PAR during relocation and contains all the information required for the supervisory functions. Figure 6-1 shows the PDR bit pattern.

6.1 ACCESS CONTROL FIELD (ACF)

This three-bit field, occupying bits 2–0 of the PDR contains the access rights to a particular page. The keys specify the manner in which a page may be accessed and whether or not a given access should result in a trap or an abort of the current operation. A memory reference which causes an abort is not completed while a reference causing a trap is completed. In the context of access control, the term "write" is used to indicate the action of any instruction which modifies the contents of any addressable word.

The keys of access control are as follows:

000	non-resident	abort all accesses
001	read-only	abort on write attempt memory management trap on read
010	read-only	abort on write attempt
011	unused	abort all accesses; reserved for future use
100	read/write	Memory Management trap upon completion of a read or write
101	read/write	Memory Management trap upon completion of a write
110	read/write	no system trap/abort action
111	unused	abort all accesses; reserved for future use

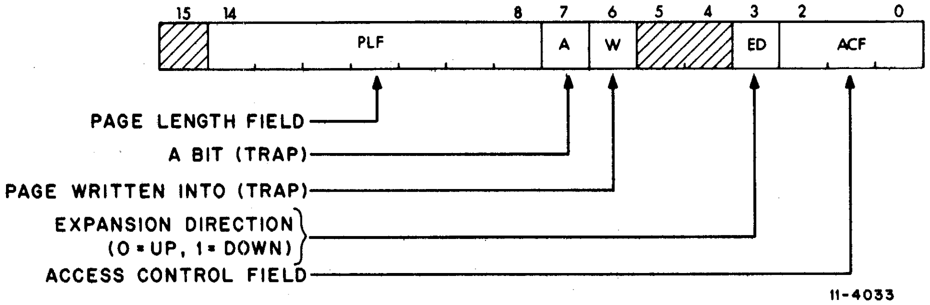


Figure 6-1 Page Descriptor Register (PDR)

It should be noted that the use of I Space in conjunction with read-only access, provides the user with a further form of protection, Execute Only.

6.2 ACCESS INFORMATION BITS (A and W)

A bit (bit 7) - This bit is used by software to determine whether or not any accesses to this page met the trap condition specified by the Access Control Field (ACF). (A = 1 is affirmative). The A bit is used in the process of gathering Memory Management statistics.

W Bit (bit 6) - This bit indicates whether or not this page has been modified (i.e., written into) since either the PAR or PDR was loaded (W = 1 is affirmative). The W bit is useful in applications which involve disk swapping and memory overlays. It is used to determine which pages have been modified and hence must be saved in their new form, and which pages have not been modified and can simply be overlaid.

The A and W bits are reset to 0 whenever either the PAR or the PDR associated with it is modified (written into) by the program, as described in Chapter 7 (Paragraph 7.2).

When the PDR (or its corresponding PAR) has just been loaded by the program, the A and W bits are 0. Refer to Figure 6-2. When the PDR is next used during relocation, its output becomes available during the BUST cycle. At T5 of this cycle (ROM OUT07=BUST) the contents of the A and W bits (SAPD+E+F RAM ATTN H and SAPD+E+F RAM WRTN INTO H) are clocked into the SAPD ATTN and SAPD WRTN INTO flip-flops. This saves the previous contents of these bits.

At T4 of the pause cycle that follows (SAPC PULSE BC9D H), if RELOC is asserted, and if the PDR is not being read or written (SCCC INT REG B L is high or not asserted) and if Memory Management is enabled (RELOC asserted), the write enable (W) input of the 3101A is enabled (SAPD WR A+W L asserted), and SAPD ATTN DATA L and SAPD WRTN DATA L are written, respectively, into the A and W bits of the selected PDR. These two gates are enabled during relocation by SCCC INT REG B L, which is high at this time.

If there is no abort condition (SSRC KT ABORT FLG L = high), and if there is a trap condition (SAPL MEM MGMT H), or if the previous contents of the A bit = 1, then SAPD ATT DATA L is asserted, and a 1 is written into the A bit of the PDR (SAPD+E+F RAM ATTN H).

Similar logic is used for SAPD WRTN DATA L, which is loaded into the W bit: if there is no abort condition, and if the cycle is a DATO, DATOB or DATIP (SAPL WRITE CYCLE H), or if the W bit was previously a 1 [SAPD WRTN INTO (1) L], then WRTN DATA is asserted.

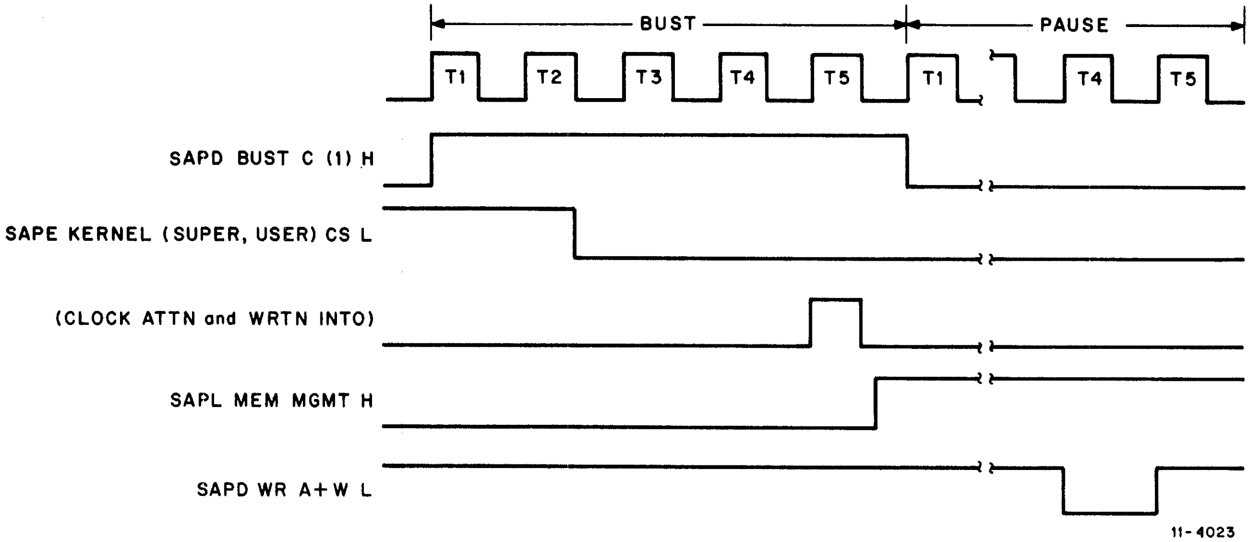


Figure 6-2 A and W Bit Timing

6.3 EXPANSION DIRECTION BIT (ED)

Bit 3 of the PDR specifies the direction in which the page is to expand. If ED = 0, the page expands upward from block number 0 to include blocks with higher addresses. If ED = 1, the page expands downward from block number 177_8 to include blocks with lower addresses.

Upward expansion is typically used for program space, and downward expansion for stack space.

6.4 PAGE LENGTH FIELD (PLF)

The seven-bit field occupying bits 14:08 of the PDR specifies the block number (BN) which defines the boundary of that page. The BN of the VA is compared against the PLF to detect length errors.

An error occurs when expanding upward if the BN is greater than the PLF, and when expanding downward if the BN is less than the PLF.

A page length error causes an abort.

6.4.1 Example of Upward Expansion

A page starting at location 00 017 000 and containing 52_8 blocks is to be defined. The page is to expand upward.

Refer to Figure 6-3. When the page expands upward, ED = 0, and the PLF is set to the number of blocks authorized for page, minus 1. As shown in the Figure:

> PLF = 51_8, which authorizes 52_8 blocks (0–51) for the page.
>
> PAF = 170_8, which establishes the physical base address = 00 017 000.
>
> PLF + PAF = 170 + 51 = 241_8, which is the PA of the last block that may be used.

Any block number [VA(12:06)] greater than 51_8 will cause an abort.

The last legal PA in this example is 00 024 176.

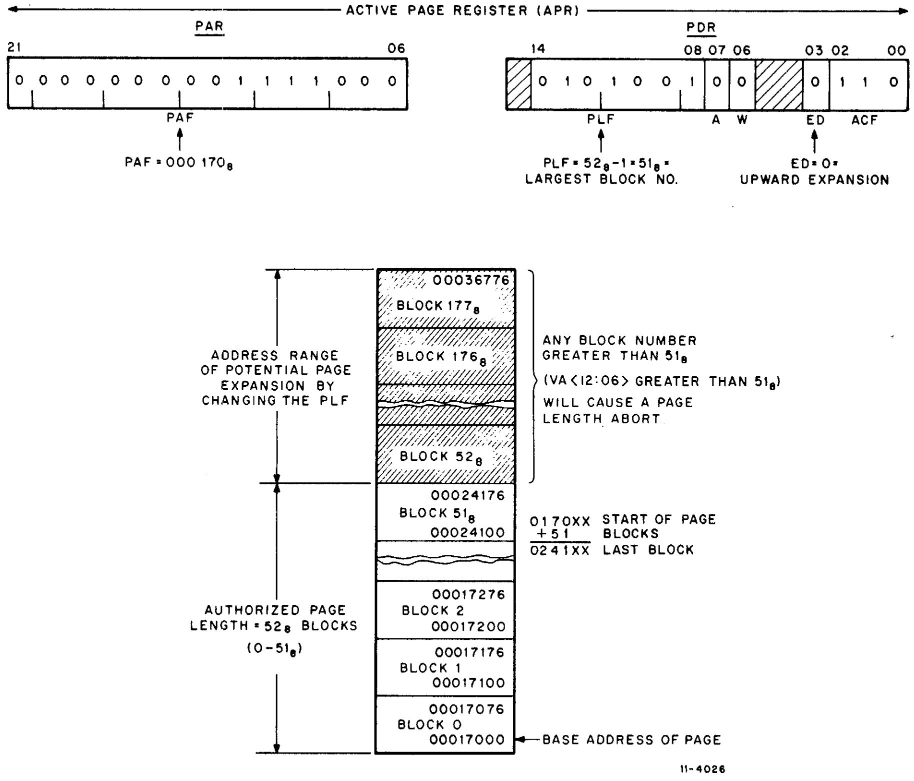


Figure 6-3 Upward Expansion

6.4.2 Example of Downward Expansion

A page whose base address is 00 017 000 is to contain a 52_8 block stack (downward expansion).

Refer to Figure 6-4. When the page expands downward, ED = 1, and the PLF is set to the 2's complement of the number of blocks authorized for the page. As shown in the Figure:

PLF = 126_8, which is the 2's complement of 52_8, the number of blocks authorized for the page.

PAF = 170_8, which establishes the physical base address of 00 017 000.

PAF + 177_8 (Maximum number of blocks per page) = 367_8, thus making the starting word address 00 036 776, and the initial setting of the stack pointer 00 037 000.

367_8 - 52_8 (or + 126_8) = 315_8, which defines the first illegal address as 00 031 576.

Another method for calculating downward expansion follows:

PLF = 126_8, which is the 2's complement of 52_8, the number of blocks authorized for the page.

PAF = 1701, which establishes the physical base address of 00 017 000.

PLF+PAF = 126+170 = 316_8, the last legal block address.

316_8+52_8 = 370_8, which is the highest legal address +2, and gives the initial setting of the stack pointers, or 00 037 000.

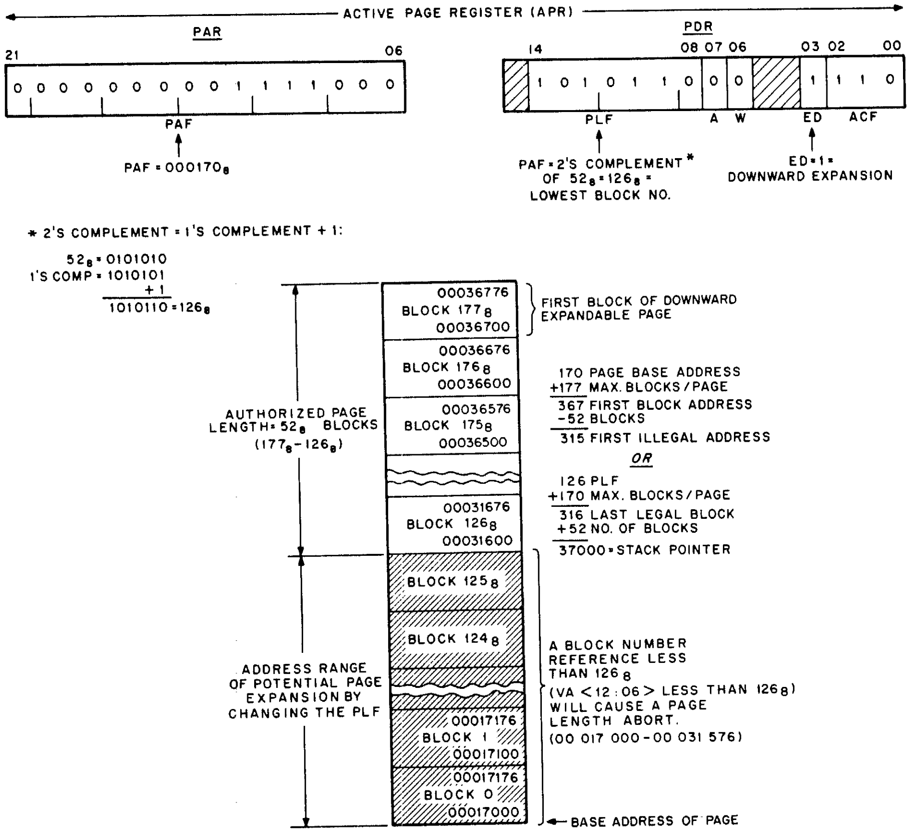


Figure 6-4 Downward Expansion

CHAPTER 7
ADDRESS DECODERS AND READING/WRITING OF PAR/PDR REGISTERS

Register addresses are decoded; the decoded signals are used as addresses in the processor as well as in Memory Management.

This chapter contains a description of the Memory Management register address decoders and of the reading and writing of the PAR/PDRs.

PARs and PDRs are loaded (written into) only under program control (with the exception of the W and A bits in the PDRs). The program may also read these registers. Both accesses are accomplished by using appropriate Unibus addresses.

7.1 REGISTER ADDRESS DECODING

Register address decoders are shown on drawings SCCB, SCCC, SCCD, SCCE, and SCCF. Addresses are buffered for various purposes on SAPH.SSRH and SCCA.

Most of the decode signals refer to the Memory Management registers, but other signals decode the address of processor registers.

Table 7-1, at the end of this paragraph, lists the signals that refer to more than one register address.

The logic on SCCB decodes all the Memory Management register addresses plus the Switch Register address:

SCCB KERNEL PDR ADRS L
17 772 300-17 772 336

SCCB KERNEL PAR ADRS L
17 772 340-17 772 376

SCCB SUPER PDR ADRS L
17 772 200-17 772 236

SCCB SUPER PAR ADRS L
17 772 240-17 772 276

SCCB USER PDR ADRS L
17 777 600-17 777 636

SCCB USER PAR ADRS L
17 777 640-17 777 676

SCCB MMR3 ADRS L
17 772 516

SCCB MMR ADRS L (MMR0,1,2)
17 777 572-17 777 576

SCCB SW REG ADRS L
17 777 570

SCCB SWR+MMR ADRS L
17 777 570-17 777 576

All these address decode signals are clocked into the flip-flops on SCCC by TIGA PSEUDO T3, which occurs approximately 30 ns after T2 of PAUSE; they are cleared by SCCD INT CLR (T1 of BUST). The Memory Management register address flip-flops are ORed to generate SCCC INT REG H.

The PAR and PDR flip-flop outputs are ORed on SCCD; SCCD APR REG H is asserted when any PAR or PDR is addressed. The unlatched version of these same signals, plus SCCB MMR3 ADRS L, SCCB SWR+MMR ADRS L and SCCE SYS INT REG L are ORed to generate SCCD INT REG ADRS H. This signal is stored in a flip-flop whose output is SCCD INTD REG (1) L. It is asserted when any one of the registers that are read out on the Internal Data Bus (INTD) is addressed.

The logic on SCCE decodes the addresses of the system registers that are located on, or read from, the SCC module (the two size registers, the ID and the Trap to 4 Error register) and the addresses of the processor Control registers (PB, PIR, SL, PS):

SCCE SYS INT REG L
17 777 760-17 777 766

SCCE SYS SIZL ADRS L
17 777 760

SCCE SYS SIZH ADRS L
17 777 762

SCCE SYS ID ADRS L
17 777 764

SCCE ERR REG ADRS L
17 777 766

SCCE INTERNAL ADRS L
17 777 770-17 777 776

SCCE PB ADRS L
17 777 770

SCCE PIR ADRS H
17 777 772

SCCE SL ADRS H
17 777 774

SCCE PS ADRS H
17 777 776

It should be noted that PSEUDO T3 is inhibited by SSRC INH PSEUDO T3 L. This signal is asserted when Memory Management is enabled and a condition exists that causes a Memory Management abort condition. INH PSEUDO T3 thus prevents changing the contents of the Memory Management registers during an abort condition caused by a reference to a Memory Management register.

SCCF GEN REG ADRS is asserted for Switch register addresses 17 777 700–17 777 717 (Console GR addresses). This signal is clocked into the SCCF GEN REG (1) H flip-flop by the LOAD ADRS switch. The flip-flop is cleared by INIT, by the CONT switch or by a LOAD ADRS to an address other than a GR address.

Table 7-1
Register Address Decode Signals

Module	Signal	Addresses Decoded
SCCB	MMR ADRS L	MMR0, MMR1, MMR2
	SWR+MMR ADRS L	MMR0 through 2, Switch Register
SCCC	INT REG H	MMR0 through 3, PARs and PDRs
SCCD	APR REG H	PARs and PDRs
	INT REG ADRS H INTD REG (1) L	All registers read on Internal Data Bus: MMR0 through MMR3, PARs and PDRs, Switch Register, System Size L and H Registers, System ID Register, Traps to 4 Error Register
SCCE	SYS INT REG L	System Size L and H, System ID, and Traps to 4 Error Registers
	INTERNAL ADDRS H	PB, PIR, SL and PS Registers
SCCF	GEN REG ADRS H GEN REG (1) H	General Register addresses from Switch Register (22-bit address).

7.2 ADDRESSING OF PAR AND PDR REGISTERS FROM THE UNIBUS

7.2.1 PAR/PDR Addresses

The Unibus addresses for the PARs and PDRs are listed in Table 7-2. The address bit configuration selects a PAR/PDR register as follows:

1. Bits (17-06) of a PAR/PDR address determine the set desired:

KERNEL:	17 772 3xx
SUPER:	17 772 2xx
USER:	17 777 6xx

2. Bit 05 of the address distinguishes between a PAR and a PDR:

PDR:	bit 05=0
PAR:	bit 05=1

3. Bit 04 defines I and D space:

I SPACE:	bit 04=0
D SPACE:	bit 04=1

4. Bits (03:01) select one of eight registers.

7.2.2 Addressing

Address Bits 0–3 – Since SCCC INT REG A L is low, the A inputs to the multiplexers on SAPK are selected. SAPK APR ADDR(3:0) and APR ADDRA(3:0) are the same as SAPH VA(04:01) B H.

PAR CHIP SELECT – The Kernel, Super and User PAR address decode signals [SCCC KERNEL PAR (1) L, SUPER (1) L and USER (1) L] are selected by the multiplexer and become the CS inputs to their respective PARs.

PDR CHIP SELECT – SCCC INT REG H is high whenever a PAR or a PDR address is decoded, and selects the A inputs to the SAPE KERNEL (or SUPER/USER) CS L multiplexer. These three signals are the chip select signals for the PDRs. A PDR is selected whenever its own address, or that of the corresponding PAR are decoded.

7.2.3 PAR/PDR Read

The outputs of the selected PAR [SAPA+B+C PAF(21:06) H] and/or PDR are input to the multiplexer on SAPM [SAPM APR BIT(15:00) H]. The PAR is selected if SAPH VA05 is high, and the PDR if VA05 is low.

SAPM APR BIT(15:00) H is in turn input to the Internal Bus data multiplexer on SSRJ.

7.2.4 PAR Write

A PAR is written at T4 of DATO or DATOB Pause cycle if TMCE KT BEND L is not asserted.

There are WRITE LOBYTE and WRITE HIBYTE write signals for each of the modes (Kernel, Super and User). Each is gated by an address decode signal [SCCC KERNEL (or SUPER/USER) PAR (1) H], by SAPK WR OK (C1=DATO or DATOB and not KT BEND), and timed by SAPC PULSE BC9B H (T4 of Pause cycle). Byte information is supplied by SAPK LO and HI BYTE B H, which are derived from the UBCB functions of the same name, which decode VA00, DATO and DATOB. The WRITE LOBYTE and WRITE HIBYTE signals are low when asserted and enable the W inputs to the 3101As.

7.2.5 PDR Write

All PDR bits, with the exception of the A and W bits (bits 07 and 06), are loaded in almost the same manner as the PAR bits. The only difference is that the gating signals include a PDR instead of a PAR address decode signal. These signals are SCCC KERNEL (or SUPER/USER) PDR (1) H.

Refer to Chapter 6 for a description of the several PDR fields. During a write operation, BR(15:08) are loaded into the Page Length Field (PLF), BR03 is loaded into the Expansion Direction Bit (ED), and BR(02:00) are loaded into the Access Control Field (ACF).

Table 7-2
PAR/PDR Unibus Addresses

Kernel

I Space			D Space		
No.	PAR	PDR	No.	PAR	PDR
0	17 772 340	17 772 300	0	17 772 360	17 772 320
1	17 772 342	17 772 302	1	17 772 362	17 772 322
2	17 772 344	17 772 304	2	17 772 364	17 772 324
3	17 772 346	17 772 306	3	17 772 366	17 772 326
4	17 772 350	17 772 310	4	17 772 370	17 772 330
5	17 772 352	17 772 312	5	17 772 372	17 772 332
6	17 772 354	17 772 314	6	17 772 374	17 772 334
7	17 772 356	17 772 316	7	17 772 376	17 772 336

Supervisor

I Space			D Space		
No.	PAR	PDR	No.	PAR	PDR
0	17 772 240	17 772 200	0	17 772 260	17 772 220
1	17 772 242	17 772 202	1	17 772 262	17 772 222
2	17 772 244	17 772 204	2	17 772 264	17 772 224
3	17 772 246	17 772 206	3	17 772 266	17 772 226
4	17 772 250	17 772 210	4	17 772 270	17 772 230
5	17 772 252	17 772 212	5	17 772 272	17 772 232
6	17 772 254	17 772 214	6	17 772 274	17 772 234
7	17 772 256	17 772 216	7	17 772 276	17 772 236

User

I Space			D Space		
No.	PAR	PDR	No.	PAR	PDR
0	17 777 640	17 777 600	0	17 777 660	17 777 620
1	17 777 642	17 777 602	1	17 777 662	17 777 622
2	17 777 644	17 777 604	2	17 777 664	17 777 624
3	17 777 646	17 777 606	3	17 777 666	17 777 626
4	17 777 650	17 777 610	4	17 777 670	17 777 630
5	17 777 652	17 777 612	5	17 777 672	17 777 632
6	17 777 654	17 777 614	6	17 777 674	17 777 634
7	17 777 656	17 777 616	7	17 777 676	17 777 636

The A and W bits are written when SAPD WR A+W L is asserted. The upper gate causes this signal to be asserted when Memory Management is enabled (RELOC) and an address other than a PAR, a PDR or MMR0 - MMR3 is being referenced. The lower gate causes WR A+W to be asserted during a write to a PAR or to a PDR.

The A and W bits are set to 0 whenever a PDR or the PAR that corresponds to it is written into:

1. SCCC INT REG B L is low during both of the above conditions. This causes SAPD ATTN DATA L (A bit) and SAPD WRTN DATA L (W bit) to be negated when the PDR or the corresponding PAR is addressed.

2. The A and W bits are written when SAPD WR A+W L is asserted. This signal is similar to the PAR and PDR write pulses, the difference being the omission of byte information and of the address decode function. SCCD APR REG H (=all PARs and PDRs) is the address decode signal in this case.

3. Thus, the A and W bits of a PDR are cleared whenever a PDR or its corresponding PAR are loaded.

CHAPTER 8
MEMORY MANAGEMENT ERROR HANDLING

Illegal memory references cause an immediate abort, i.e., the memory reference is not completed and an interrupt is sent to the processor. It should be noted that the instruction containing the aborted memory reference is not completed.

Refer to Figure 8-1. Three kinds of faults cause an abort: page length violation, non-resident memory and read only violation.

A length fault is caused by a memory reference outside the limits set by the Page Length Field and ED bit of the PDR. It is explained in Paragraph 8.1.

A non-resident fault is caused by an attempted access to a prohibited page. A read-only fault is caused by an attempted write to a page for which only read accesses are allowed. The Access Control Field (ACF) of the PDR determines the allowable

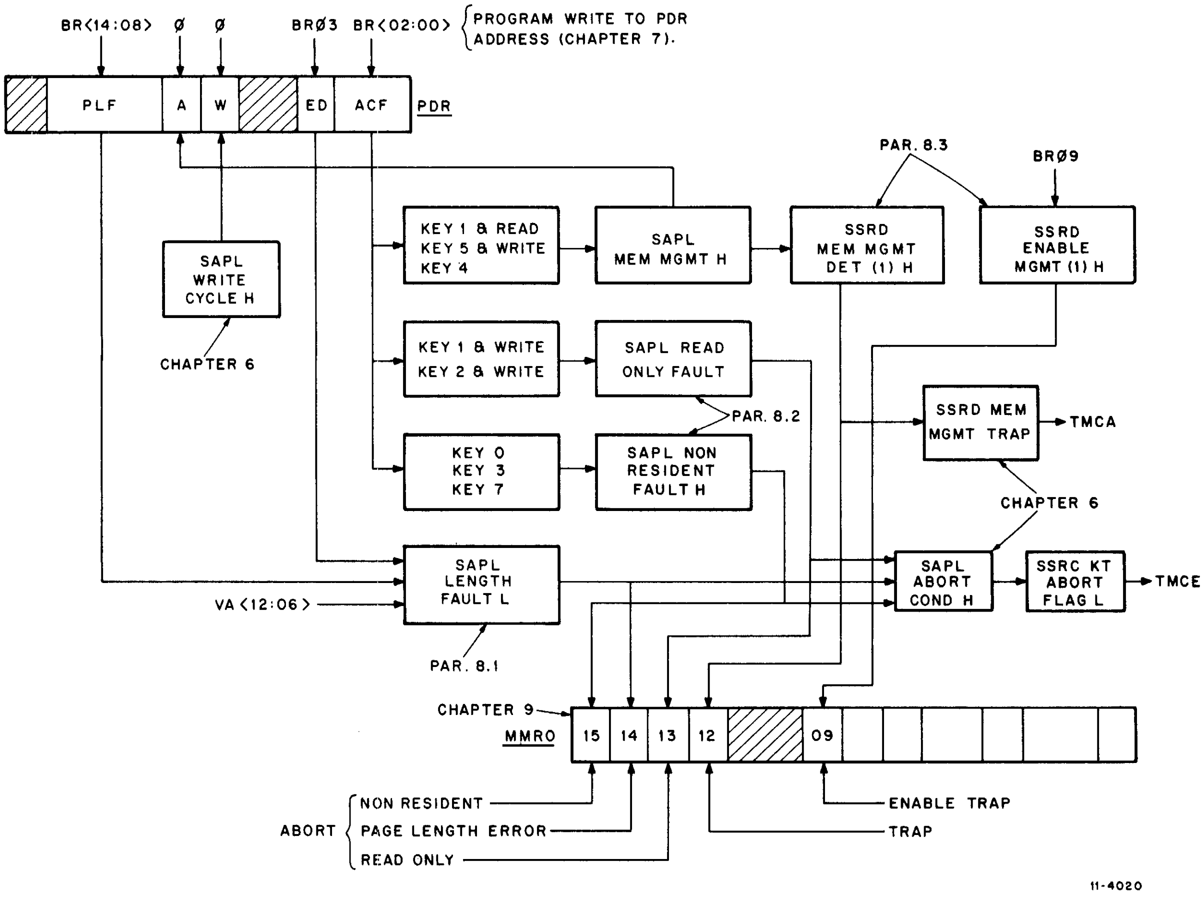


Figure 8-1 Traps and Aborts

access modes of a given page. These two faults are described in Paragraph 8.2.

Paragraph 8.3 explains the Memory Management traps. Certain access keys in the ACF cause a trap instead of an abort. A trap is only executed at the end of the instruction, and the memory reference causing the trap is executed.

Finally, the A and W bits are set in the PDR to aid in statistics gathering by the executive program. The A bit is set every time that access to the selected page results in a trap condition; the W bit is set if the page is written into (modified). This mechanism is explained in Chapter 6.

Information on aborts and traps is stored in bits 15:09 of MMR0. Additional information, to help the program determine the origins of aborts and traps, is stored in the remaining bits of MMR0 as well as in MMR1, MMR2, and MMR3. Chapter 9 describes these registers.

All aborts and traps generated by Memory Management are vectored through Kernel space address 250.

8.1 PAGE LENGTH ABORTS

When Memory Management is enabled, i.e., in 18- or 22-bit mapping, the VA is examined to determine whether it falls within the selected page. If it does not, the VA is illegal and the instruction is aborted. An illegal processor mode [PS(15:14)=10] also causes a page length abort.

8.1.1 Length Fault

SAPL LENGTH FAULT L and MM LENGTH FAULT H are asserted when VA(12:06) (block number) is greater than the PLF and the expansion is upward (SAPL PGE EXPN DOWN L is not asserted, or ED = 0), or when VA(12:00) (block number) is less than the PLF and the expansion is downward (SAPD+E+F PGE EXPN DOWN H is asserted, or ED = 1).

SAPL LENGTH FAULT L is stored in bit 14 of MMR0 (SSRC MMR0 BIT 14). (Refer to Chapter 9).

LENGTH FAULT is also ORed with the read-only and non-resident faults to generate SAPL ABORT COND H. This function in turn generates SSRC KT ABORT FLAG L, which notifies the processor abort logic of the abort condition. (Refer to Paragraph 8.2).

8.1.2 Illegal Processor Mode

A length fault occurs if the PSW contains an illegal processor mode [PS(15:14) = 10].

If this is the case, none of the mode flip-flops on SSRB are selected, and no PAR/PDR set is selected, since the mode generates the Chip Select input to these registers (refer to Chapter 3). This causes the output of the PAR to be all ones, SAPD PGE EXPN DOWN H to be high, and LENGTH FAULT to be asserted.

8.2 ACCESS CONTROL FIELD ABORTS AND TRAPS

8.2.1 Non-Resident and Read-Only Protection

A Page Descriptor Register (PDR) is selected in the same manner as a Page Address Register (PAR). After the selection occurs, three bits from the PDR are decoded as an access key. If the access rights designated by the key are inconsistent with the current memory reference, the memory reference is not completed and an abort to Kernel space 250 occurs.

When the access key is set to 0, the page is defined as non-resident, and an abort prevents any attempt by a program to access a non-resident page. Using this feature to provide memory protection, only those pages associated with the current program are set to legal access keys. The access control keys of all other program pages are set to 0, which prevents illegal memory references.

The access control key for a page can be set to 2, allowing read memory references to the page, but aborting any attempt to write into the page. This read-only type of memory protection can be given to pages that contain common data, subroutines, or shaded algorithms. This type of memory protection makes certain that access rights to a given information module are user-dependent, i.e., the access right to a given information module may be varied for different users by altering the access control key.

A PAR in each of the sets (Kernel, User, and Supervisor modes) may be set up to reference the same physical page in memory and each may be keyed for different access rights. For example, the User access control key might be 2 (read-only access), the Supervisor access control key might be 0 (non-resident), and the Kernel access control key might be 6 (allowing completer read/write access).

8.2.2 Access Faults (Aborts)

The Access Fault (ACF) is decoded on SAPL to detect abort and trap conditions.

Non-resident (key = 0) and unused keys (3 or 7) cause SAPL NON RES FAULT L to be asserted and stored in bit 15 of MMR0 (SSRC MMR0 BIT 15).

If PS(15:14) contain 10 (illegal mode), none of the mode flip-flops on SSRB are selected, and no PAR/PDR set is selected, since the mode generates the Chip Select input to these registers (refer to Chapter 3). This causes the output of the PDR to be all 1s. The ACF is thus 7 and SAPL NON RES FAULT L is asserted.

SAPL WRITE CYCLE H is asserted when the bus cycle is either a DATO, a DATOB, or a DATIP. It is used to detect all write or read/modify/write (DATIP followed by DATO or DATOB) cycles.

WRITE CYCLE is ANDed with both abort-on-write keys (1 and 2) to generate SAPL READ ONLY FAULT, which is stored in bit 13 of MMR0 (SSRC MMR0 BIT 13).

The non-resident and read-only fault decoders are ORed with SAPL LENGTH FAULT (refer to Paragraph 8.1) to generate SAPL ABORT COND H.

8.2.3 Abort Flag

SAPL ABORT COND H asserts SSRC KT ABORT FLG L (which informs the abort logic on TMCE that a Memory Management abort condition has been detected) if RELOC is asserted, and if the cycle is not a BEND.

KT ABORT FLG is the input to the SSRC ABT FLG flip-flop, which is clocked by SSRK PULSE BC89 H (=TS3 of Pause cycle) and latches SSRC KT ABORT FLG L. The flip-flop is cleared by UBCB ABORT ACKN H.

8.3 MEMORY MANAGEMENT TRAPS

A timeshared system swaps programs, or parts of programs, in and out of memory using secondary storage facilities such as disk systems. In a swapping environment, the operating system must provide the software routines that decide which programs should be swapped and when and how these programs can be swapped between memory and secondary storage.

The operating system routines can be simple or complex depending on system requirements, e.g., the amount of overhead time that can be tolerated. The operating system may also have to decide which active page is least likely to be required in the immediate future and may therefore be swapped out to make memory space available for a new program.

To make such a Memory Management decision, the operating system requires statistics on the use of active pages. Some indication of whether a program has been modified during its residence in memory is also desirable. If it has been modified, the program must be swapped (rewritten) into secondary storage. If no modification has been made, and the program can always be recalled from secondary storage, the space it occupies in memory can be overlayed, thus eliminating the swapping delay.

The logic provides the kind of information required by an operating system to gather Memory Management statistics on the use of active pages. The availability of this information in the hardware reduces the overhead time of any routine, simple or complex, in the efficient management of memory.

The Page Descriptor Register associated with each active page includes a W (written into) and an A (attention) bit. When any active page is written into, the W bit is set by the logic; therefore, by testing the W bit, the Memory Management software routine can decide whether a page can be overlayed or if it needs to be swapped out (e.g., copied onto a disk).

The A bit has several uses. To use this feature, the system programmer may enable the Memory Management trap logic. He then sets the access control keys of the active pages of interest for special trap conditions. Access control keys are provided to cause:

1. Memory Management trap on read (including instruction fetch)
2. Memory Management trap on write
3. Memory Management trap on read or on write.

The A bit for the active page is then set when the page is accessed and a Memory Management trap condition occurs. The vector at trap location 250 Kernel address space causes the operating system routine to service the Memory Management trap. The routine can test the A bit to accumulate statistics on the use of that page. When a swapping decision is required of the operating system, these statistics can be examined to determine the more active pages (which might therefore be retained in memory).

Access Control Traps
Keys 1, 4, and 5 of the ACF are examined to determine whether a trap condition exists.. The following functions are generated:

SAPL KEY=1.WRI L= key 1 during a write cycle (DATO, DATOB or DATIP).

SAPL KEY=4 L = key 4 on read or write.

SAPL KEY=5WR L = key 5 during a write cycle.

These three functions are ORed to generate SAPL MEM MGMT H which is stable by the end of T5 of a BUST cycle. If Memory Management is enabled (SSRE RELOC H) and if the address is not a Memory Management register (SCCC INT REG L), then SSRD CLK TRAP H is asserted.

Refer to Figure 8-2. If bit 9 of MMR0 [SSRD ENABLE MGMT (1) H] is set, and if the abort flag is not set, and if SSRD MGT TP DET DLY L is not asserted (see below), SSRD MEM MGMT TRAP L is asserted via its "pre-Mem Management Trap" gate. This occurs during a Pause cycle, and if TMCE BRQ CLK H is asserted (at TS3) during this cycle, MEM MGMT TRAP is clocked into the priority flip-flop on TMCA.

At SSRK PULSE BC89 H (which occurs at TS3 of every Pause cycle), and if there are no abort conditions, SSRD MEM MGMT B L is asserted and sets SSRD MEM MGMT DET (1) H, which is bit 12 of MMR0. At the same time, if MMR0 bit 9 is set, and if SSRD MGT TP DET DLY L is not asserted, SSRD MGMT HOLD L sets the latch flip-flop. This flip-flop keeps SSRD MEM MGMT TRAP L asserted; it is cleared when the trap is acknowledged by the processor trap logic or until a Memory Management abort is detected.

This hold flip-flop is necessary because, 50 ns after SSRD MEM MGMT DET is set, SSRD MGT TP DET DLY L is asserted and disables the "pre-Mem Managment Trap" input gate to SSRD MEM MGMT TRAP L. This gate stays disabled until the program clears the MEM MGMT DET flip-flop (MMR0 bit 12) by writing a 0 into it. The A and W bits are clocked at T4 of Pause (PULSE BC9D).

The logic thus ensures that only one trap can be sensed per instruction, and that no subsequent trap can be clocked by the TMCA logic until the program has reset bit 12 of MMR0.

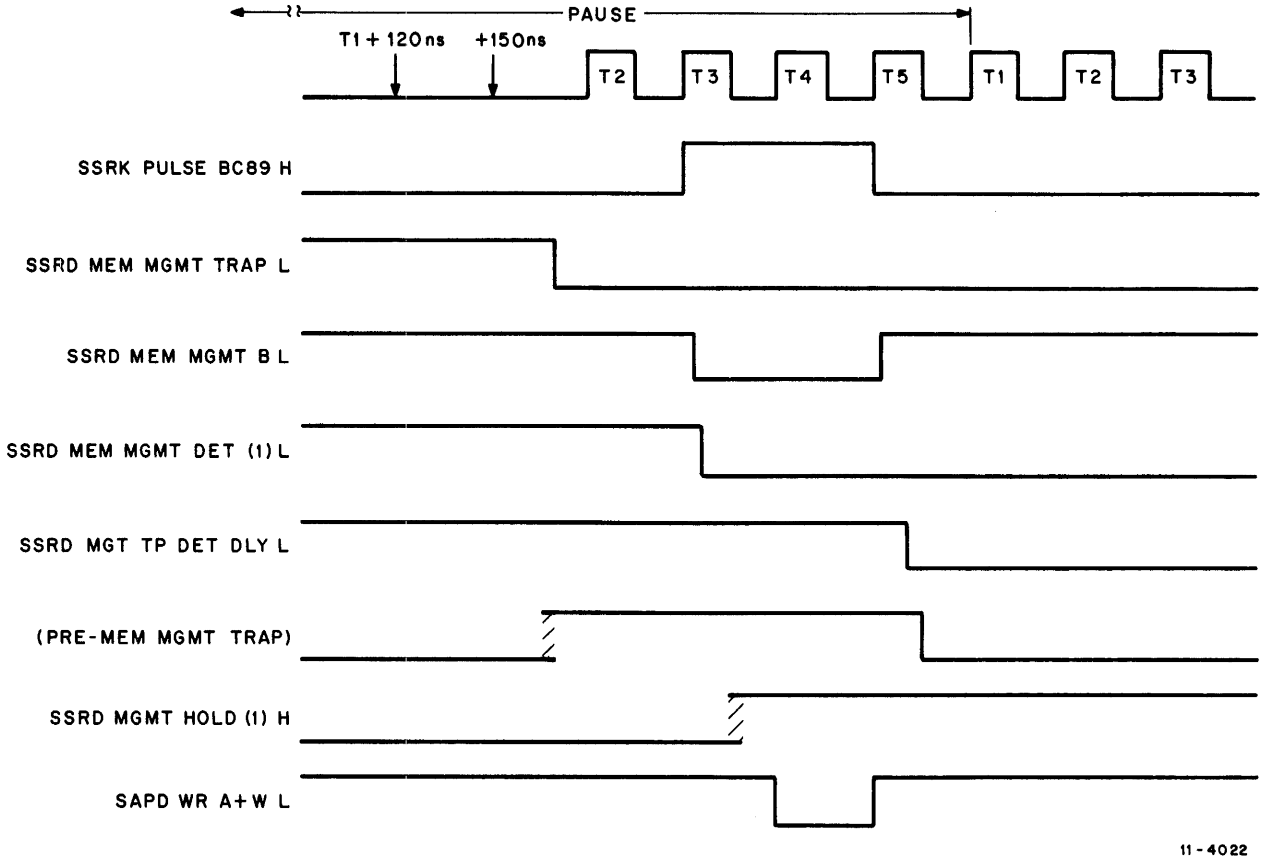


Figure 8-2 Trap Timing

CHAPTER 9
MEMORY MANAGEMENT REGISTERS (MMR0, 1, 2, AND 3)

Aborts and traps generated by the Memory Management hardware are vectored through Kernel virtual location 250. Memory Management Registers 0, 1, 2, 3 are used in order to distinguish an abort from a trap, to determine why the abort or trap occurred, and to allow for easy program restarting. Note that an abort or trap to a location which is itself an invalid address will cause an-other abort or trap. Thus, the Kernel program must ensure that Kernel VA 250 is mapped into a valid address, or a loop will occur which will require console intervention.

9.1 MMR0

MMR0 contains error flags, the page number whose reference caused the abort, and various other status flags. The register is organized as shown in Figure 9-1. Its address is 17 777 572.

This paragraph first defines the meaning of the various bits in MMR0, then the logic that controls these bits.

Setting bit 0 of this register enables address relocation and error detection. This means that the bits in MMR0 become meaningful.

Bits 15–12 are the error flags. They may be considered to be in a "priority queue" in that "flags to the right" are less significant and should be ignored if more than one of them is set; i.e., a "non-resident" fault service routine would ignore length, access control, and Memory Management flags. A "page length" service routine would ignore access control and Memory Management faults, etc.

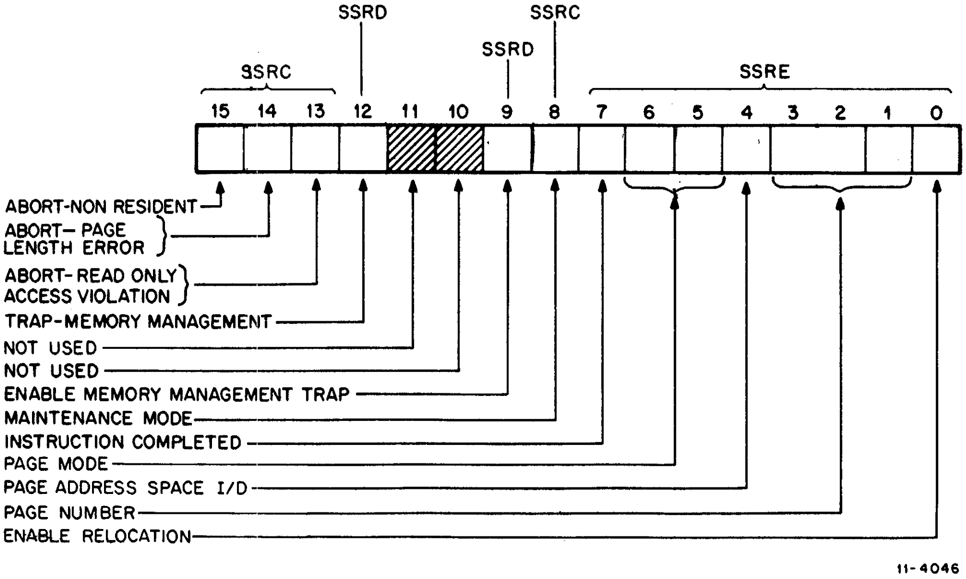


Figure 9-1 MMR0

Bits 15–13 when set (error conditions) cause Memory Management to freeze the contents of bits 1–7 and of Registers 1 and 2. This is done to facilitate error recovery.

These bits may also be written under program control. No abort will occur, but the contents of the Memory Management registers will be locked up as in an abort.

Bits 15–12 are enabled by SSRE RELOC. RELOC is true when an address is being relocated by the Memory Management unit. This implies that either MMR0, bit 0 is equal to 1 (Relocation operating) or that MMR0, bit 8 (Maintenance) is equal to 1 and the memory reference is the final one of a destination calculation (Maintenance/Destination mode).

9.1.1 Aborts

Bit 15 is the "Abort-Non-Resident" bit. It is set by attempting to access a page with an Access Control Field (ACF) key equal to 0, 3, or 7. It is also set by attempting to use Memory Relocation with a processor mode of 2. [PS(15:14)=10].

Bit 14 is the "Abort-Page-Length" bit. It is set by attempting to access a location in a page with a block number (VA bits, 12–6) that is outside the area authorized by the Page Length Field (PLF) of the Page Descriptor Register (PDR) for that page. Bit 14 is also set by attempting to use Memory Relocation with a processor mode of 2.

Bit 13 is the "Abort-Read-Only" bit. It is set by attempting to write in a "Read-Only" page. "Read-Only" pages have access keys of 1 or 2.

The logic that generates these aborts is explained in Chapters 8 and 9.

9.1.2 Traps and Trap Enable

Bit 12 is the "Trap-Memory Management" bit. It is set whenever a Memory Management trap occurs; that is, a read operation which references a page with an Access Control Field (ACF) of 1 or 4, or by a write operation to a page with an ACF key of 4 or 5.

Bits 11 and 10 are spares. They are always read as 0, and should never be written. They are unused and reserved for possible future expansion.

Bit 9 is the "Enable Memory Management Traps" bit. It is set or cleared by doing a direct-write into MMR0. If bit 9 is 0, no Memory Management traps will occur. The A and W bits will, however, continue to log potential Memory Management trap conditions. When bit 9 is set to 1, the next Memory Management trap condition will cause a trap, vectored through Kernel VA 250.

Note that if an instruction which sets bit 9 to 0 (disable Memory Management Trap) causes a Memory Management trap condition in any of its memory references, prior to and including the one actually changing MMR0, then the trap will occur at the end of the instruction.

The trap logic is described in Chapter 8.

9.1.3 Maintenance/Destination Mode

Bit 8 specifies that only Destination mode references will be relocated using Memory Management. This mode is only used for maintenance purposes. Refer to Chapter 4.

9.1.4 Instruction Complete

Bit 7 indicates that the current instruction has been completed. It will be set to 1 during T bit, Parity, Odd Address, and Time Out traps and interrupts. This provides error-handling routines with a way of determining whether the last instruction will have to be repeated in the course of an error recovery attempt (after an abort). Bit 7 is Read-Only (it cannot be written). Note that EMT, TRAP, BPT, and IOT do not set bit 7.

Bit 7 [SSRE INSTR COMP (1) H] is set by SSRA BRK.30 (1) H if there has been no previous Memory Management abort condition. [SSRC ND ERROR (1) H = none of bits 15:13], BRK.30 is asserted at SSRK PULSE23 H (TS2) when the output of the Memory Management ROM bits 03:00 equal 100; this occurs during the BRK.30 cycle (Flows 12).

Bit 7 is cleared, if there is no abort, at TS3, when a new instruction is fetched (SSRH LOAD IR H = UIRK asserted).

The following conditions may occur during the course of program execution:

1. If the first abort is a Memory Management abort, NO ERROR clears at T4 of PAUSE, before entry into the Service Flows. Therefore, INSTRUCTION COMPLETE is not set by BRK.30.

2. If the first (and only) abort is not a Memory Management abort, INSTRUCTION COMPLETE is set by BRK.30, but is cleared by FET.00 (217) at the beginning of the instruction fetch sequence (after the pushes to the stack have been successfully executed).

3. If the first abort is not a Memory Management abort, then INSTRUCTION COMPLETE is set at BRK.30 (because NO ERROR is set). If a Memory Management abort then occurs during the Service Flows, NO ERROR is cleared at T4 of PAUSE. This prevents INSTRUCTION COMPLETE from being cleared until the abort condition bits in MMR0 are cleared. In this case, MMR2 contains either a vector address or the stack pointer, and not a program address.

9.1.5 Processor Mode

Bits 5, 6 indicate the CPU mode (User/Supervisor/Kernel) associated with the page causing the abort. (Kernel = 00, Supervisor = 01, User = 11.) If an illegal mode (10) is specified, bits 15 and 14 will be set and an abort occurs.

Bits 05 and 06 of MMR0 show the actual Processor mode [as decoded by SSRB MMR0 MODE0 (and 1) H from the mode flip-flops]. The actual mode may not be the same as that shown by PS(15:14). Refer to Chapter 3.

9.1.6 Address Space and Page Number

Bit 4 indicates the type of address space (I or D) the unit was in when an abort occurred (0 = I Space, 1 = D Space). It is used in conjunction with bits 3-1, Page Number.

Bits 3:1 contain the page number of a reference causing a Memory Management fault. Note that pages, like blocks, are numbered from 0 upwards.

SAPK IND DATA H is stored in bit 4 of MMR0. The SSRB space flip-flop that is asserted during a given cycle gates the I or D Space information from MMR3 [SAPK D S K (or S/U), H] to generate this signal. The SAPK D S signals are described in Chapter 3.

Bits 03:01 of MMR0 are loaded with VA(15:13), which give the address of the selected PAR/PDR set.

9.1.7 Enable Relocation

Bit 0 is the "Enable Relocation" bit. When it is set to 1, all addresses are relocated by the unit. When bit 0 is set to 0, the Memory Management Unit is inoperative and addresses are not relocated or protected. Chapter 4 explains the logic associated with this bit.

9.1.8 Read/Write Under Program Control

MMR0 is read by the processor on the Internal Data Bus through the multiplexer on SSRJ. Refer to Section II, Chapter 2.

Bits 00, 08, 09, and 12:15 can be written into by the program from the BR.

Refer to Figures 9-2 and 9-3. SCCB MMR ADRS L decodes the addresses of MMR0 - MMR2. At TIGA PSEUDO T3 H, it is clocked into the SCCC SSR REG flip-flop, whose output is ANDed with TMCE C1 H to generate SCCC WRITE MMR0 REG H [MMR0 is the only writable register of the three (MMR0-2)]. Since the address of MMR0 - MMR2 differ only by bits 02 and 01 of the VA, this signal, when NANDed with VA(02:01), indicates a write to MMR0, and selects the A inputs [BR(15:13)] to the multiplexer input to the SSRC MMR0 BIT(15:13) flip-flop.

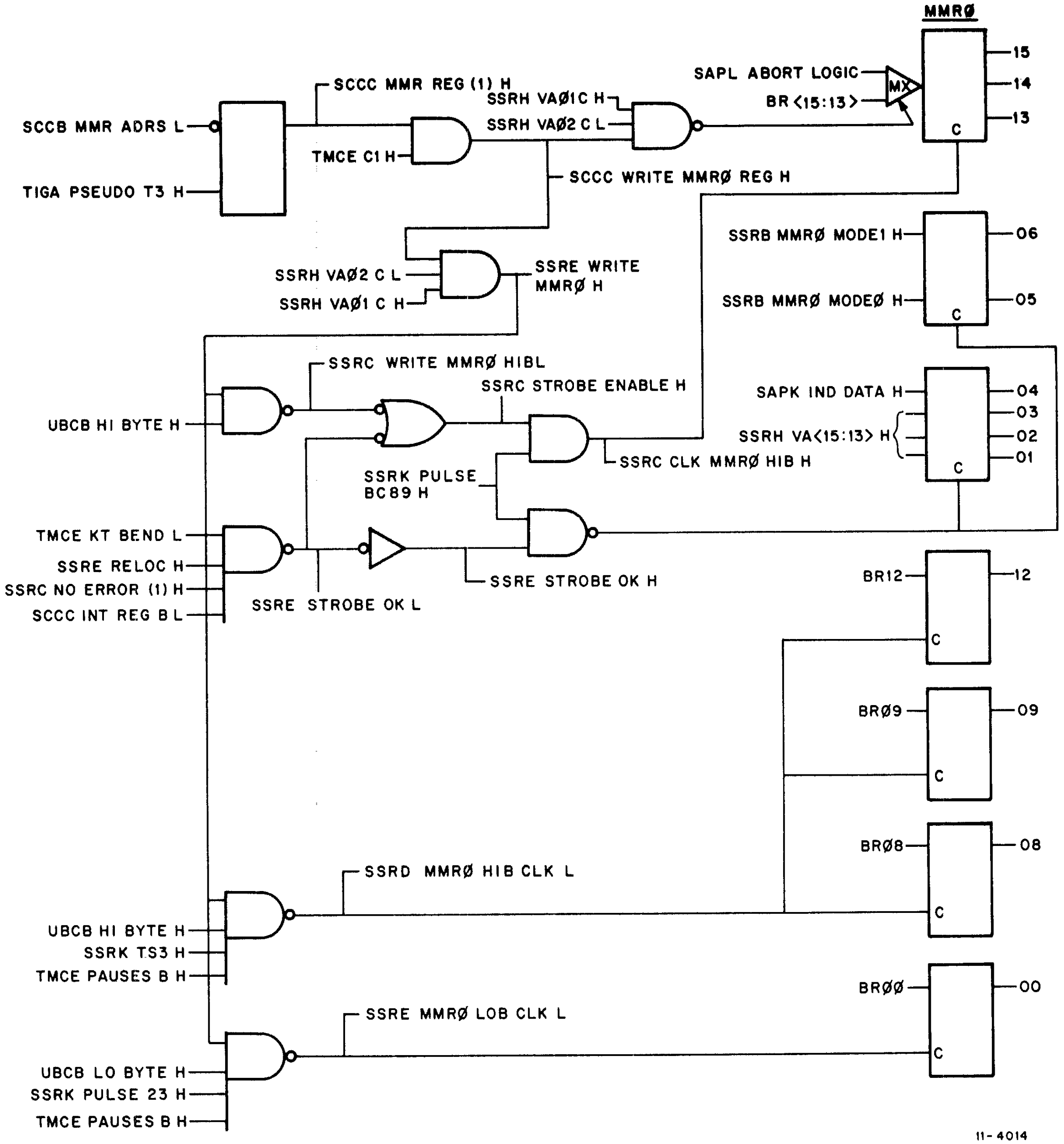


Figure 9-2 Clocking of MMR0

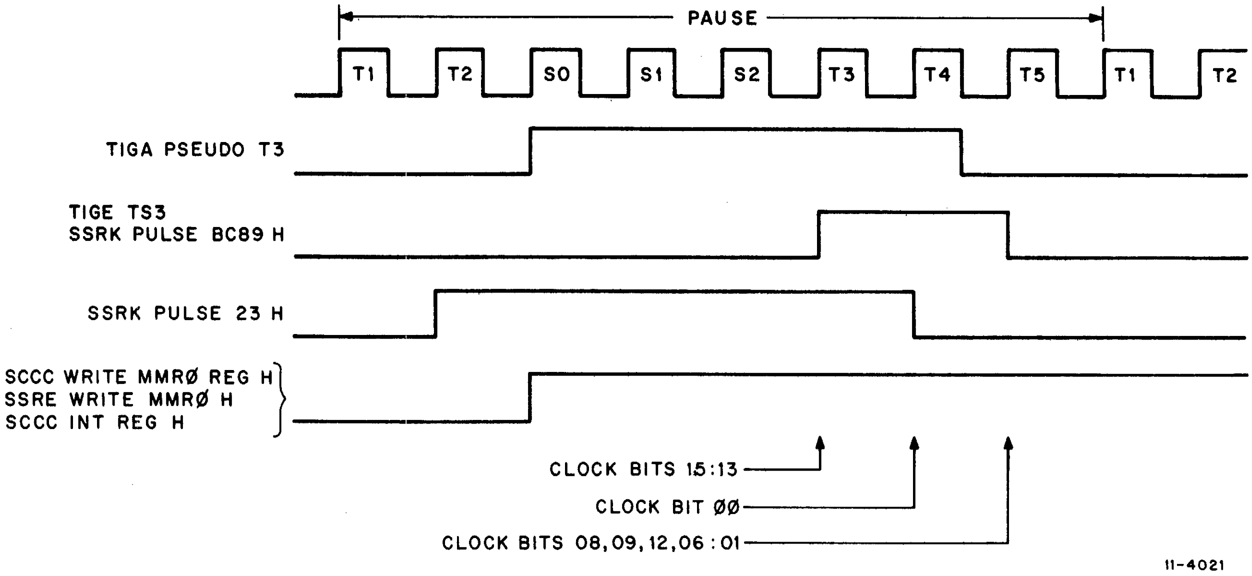


Figure 9-3 MMR0 Write Timing

SSRE WRITE MMR0 H is the same as the multiplexer select signal, but not inverted.

1. Gated with UBCB HI BYTE H and SSRK PULSE BC89 H (TS3 of PAUSE), it clocks the output of the multiplexer [SSRC PRE MMR0 BIT(15:13) H] into MMR0 bit 15:13 at T3.
2. NANDed with UBCB HI BYTE H, TMCE PAUSES B H and SSRK TS3 H, it clocks BR12, 09 and 08 into the corresponding bits of MMR0 at T5.
3. NANDed with UBCB LO BYTE H, TMCE PAUSES B H and SSRK PULSE 23 H (TS2), it clocks BR00 into bit 00 of MMR0 at T4.

9.1.9 Bits Controlled by Memory Management

Bits 15:13 are also clocked automatically on abort conditions. Bits 06:01 are clocked on every memory reference, but cannot be changed once an abort bit (15:13) is set.

SSRE STROBE OK is asserted when Memory Management is enabled (SSRE RELOC is asserted), if there has been no previous Memory Management abort condition (SSRC NO ERROR), if no Memory Management register is being read or written (SCCC INT REG B L not asserted), and if TMCE KT BEND is not asserted.

SSRE STROBE OK is gated with SSRK PULSE BC89 H (TS3).

1. On the leading edge of the pulse, at T3, the abort bits from SAPL are gated into bits 15:13 of MMR0 (since the register is not being read or written, the multiplexer select signal is high, and the B inputs are selected).
2. On the trailing edge of the pulse, at T5, bits 06 - 01 are clocked into MMR0.

9.2 MMR1

MMR1 records any autoincrement/decrement of the general-purpose registers. MMR1 is cleared at the beginning of each instruction fetch if no abort condition is present. Whenever a general-purpose register is either autoincremented or autodecremented, the register number and the amount (in 2's complement notation) by which the register was modified, is written into MMR1.

The information contained in MMR1 is necessary to accomplish an effective recovery from an error resulting in an abort. The low order byte is written first and it is not possible for a PDP-11 instruction to autoincrement/decrement more than two general-purpose registers per instruction (refer to Section II, Chapter 1). Only three bits are available to record the register number; thus, it is up to the software to determine which set of registers (User/Supervisor/Kernel–General Set 0/General Set 1) was modified, by determining the CPU and Register modes at the time of the abort. The 6-bit displacement on R6 (SP) that can be caused by the MARK instruction cannot occur if the instruction is aborted.

MMR1 is read on the Internal Data Bus through the multiplexer on SSRJ. Its address is 17 777 574. Refer to Section II, Chapter 2.

Figure 9-4 shows the format of MMR1. Its logic is on drawing SSRF.

1. The register number is taken from the General Register address bits, GRAC GRA(3:0) L and are encoded to fall in the range of 0 – 7. Refer to Section II, Chapter 2 of this manual (Data Paths).

2. The amount of the increment or decrement is that shown by the KOMX multiplexer input to the ALU. This logic is described in the same chapter as the General Register address bits. The multiplexer to which they are input selects the complement of the KOMX if the cycle calls for a decrement.

3. Bits 03:00 of the Memory Management ROM informs the MMR1 logic of the autoincrement or decrement. A decrement causes a ROM output of 011, an increment an output of 010. SSRA ONE CHANGED (1) H is asserted when the ROM output is either 010 or 011. SSRA AUTO DEC is asserted when the output is 011. This signal supplies the sign bit to the increment/decrement value.

4. A Memory Management abort [SSRC NO ERROR (1) H not asserted], latches the contents of MMR1.

MMR1 and ONE AUTOED are both cleared, if no previous Memory Management aborts have occurred (NO ERROR),

1. during an instruction fetch (SSRH LOAD IR), or

2. during the Service Flows (SSRA BRK.30), or

3. if INIT is asserted.

For the first increment or decrement, SSRF ONE AUTOED (0) H is high (the flip-flop is cleared). At T5B, if there is a change to the register, and if SSRA ONE CHANGED (1) H is asserted, the change to the register and the register number are written into the low order byte of MMR1 (bits 07:00).

The ONE AUTOED flip-flop is then clocked by the trailing edge of T5 and its (1) output goes high.

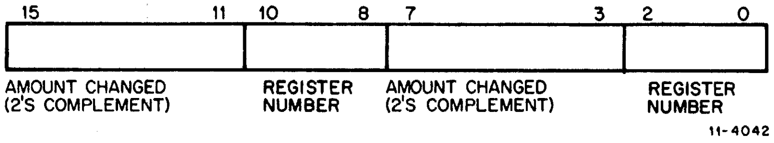

15
11
10
8
7
3
2
0
AMOUNT CHANGED (2'S COMPLEMENT)
REGISTER NUMBER
AMOUNT CHANGED (2'S COMPLEMENT)
REGISTER NUMBER
11-4042

Figure 9-4 MMR1

Thus, if there is a change to another register during the same instruction, it will be stored into the high order byte (bits 15:08) of MMR1.

9.3 MMR2

MMR2 is the VA Program Counter. (Refer to Figure 9-5.) It is loaded with the 16-bit VA at the beginning of each instruction fetch, or with the address Trap Vector at the beginning of an interrupt, "T" Bit trap, Parity, Odd Address, and Timeout traps. Note that MMR2 does not get the Trap Vector on EMT, TRAP, BPT and IOT instructions. MMR2 is Read-Only; it cannot be written.

MMR2 is loaded at TS4, if there has been no previous Memory Management abort, when the IR is loaded during instruction fetch, or during the BRK.30 cycle. Note that the EMT, TRAP, BPT and IOT instructions do not use the BRK.30 cycle.

MMR2 is shown on drawing SSRH. Its address is 17 777 576. It is read on the Internal Data Bus through the multiplexer on SSRJ. Refer to Section II, Chapter 2.

9.4 CLEARING STATUS REGISTERS FOLLOWING TRAP/ABORT

At the end of a fault, service routine bits 15–12 of MMR0 must be cleared (set to 0) to resume error checking. On the next memory reference following the clearing of these bits, the various registers will resume monitoring the status of the addressing operations. MMR2 will be loaded with the next instruction address. MMR1 will store register change information and MMR0 will log Memory Management status information.

9.5 MULTIPLE FAULTS

Once an abort has occured, any subsequent errors that occur will not affect the state of the machine. The information saved in MMR0 – MMR2 will always refer to the first abort that it detected. However, when multiple traps occur, the information saved will refer to the most recent trap that occurred.

In the case that an abort occurs after a trap, but in the same instruction, only one stack operation will occur; and the PC and PS at the time of the abort will be saved.

9.6 MMR3

Refer to Figure 9-6. MMR3 enables or disables:

1. The use of the D space PARs and PDRs,
2. 22-bit mapping,
3. Unibus Map mapping.

When D space is disabled, all references use the I space registers; when D space is enabled, both the I space and D space registers are used. Bit 0 refers to the User's Registers, bit 1 to the Supervisor's and bit 2 to the Kernel's. When the appropriate bits are set, D space is enabled; when cleared, it is disabled. Bit 03 is read as zero and never written; it is reserved for future use. Bit 04 enables 22-bit mapping. If Memory Management is not enabled, bit 04 is ignored and 16-bit mapping is used.

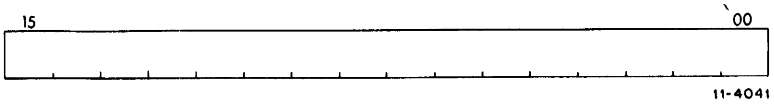


Figure 9-5 MMR2

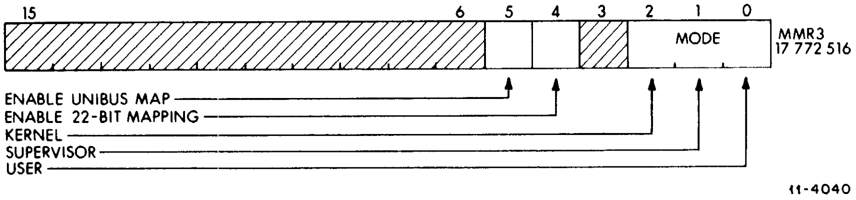


Figure 9-6 MMR3

If bit 4 is clear and Memory Management is enabled (bit 0 of MMRO is set), the computer uses 18-bit mapping. If bit 4 is set and Memory Management is enabled, the computer uses 22-bit mapping. Bit 5 is set to enable relocation in the Unibus Map; the bit is cleared to disable relocation. Bits 6-15 are unused. On initialization this register is set to 0 and only I space is in use.

The following table is a summary of these conditions:

Bit	State	Operation
5	0	Unibus Map relocation disabled
	1	Unibus Map relocation enabled
4	0	Enable 18-bit mapping if bit 0 of MMRO is set
	1	Enable 22-bit mapping if bit 0 of MMRO is set
2	1	Enable Kernel D Space
1	1	Enable Supervisor D Space
0	1	Enable User D Space

MMR3 is loaded from BR(05:00) by SCCL MMR3 CLK L [=T4+15 ns of PAUSE during a write cycle and the address decode, SCCC MMR3 (1) H].

MMR3 is shown on drawing SCCL. Its address is 17 772 516. It is read on the Internal Data Bus through the multiplexer on SCCH. Refer to Section II, Chapter 2.

SECTION V

UNIBUS MAP

Unless otherwise indicated, references within this section pertain to this section only.

SECTION V UNIBUS MAP
CONTENTS

Page

ILLUSTRATIONS

Figure No. Title Page

TABLES

Table No. Title Page

INTRODUCTION

INTRODUCTION

The Unibus Map is the interface between the Unibus and Cache. It responds as a slave device to Unibus signals and converts 18-bit Unibus addresses to 22-bit Cache addresses.

The reader of this section should be familiar with the concepts related to PDP-11/70 Address Space. The Introduction to Section IV of this manual (Memory Management) describes PDP-11/70 Address Space in detail.

The top 4K word addresses of the 128K Unibus addresses are reserved for CPU and I/O registers and are called the Peripherals Page (see Figure I-1). The lower 124K addresses are used by the Unibus Map to reference physical memory.

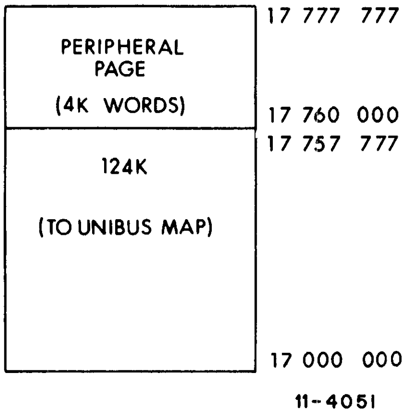


Figure I-1 Unibus Address Space

The Unibus Map is the interface to memory from the Unibus. The operation is transparent to the user, if it is disabled.

Relocation Disabled

If the Unibus Map relocation is not enabled, an incoming 18-bit Unibus address has 4 leading zeros added for referencing a 22-bit Physical Address (PA). The lower 18 bits are the same. No relocation is performed.

Relocation Enabled

There are a total of 31 mapping registers for address relocation. Each register is composed of a double 16-bit PDP-11 word (in consecutive locations) that holds the 22-bit base address. These registers have Unibus addresses in the range 17 770 200–17 770 372.

If Unibus Map relocation is enabled, the 5 high order bits of the Unibus address are used to select one of 31 mapping registers. The low order 13 bits of the incoming address are used as an offset from the base address contained in the 22-bit mapping register. To form the PA, the 13 low order bits of the Unibus address are added to 22 bits of the selected mapping register to produce the 22-bit PA. The lowest order bit of all mapping registers is always a zero, since relocation is always on word boundaries.

The Unibus Map is disabled upon the occurrence of any of the following:

1. Power-up
2. Depressing the START switch on the Console, and
3. The execution of a RESET instruction.

These all cause the assertion of INIT, which clears MMR3. It should be noted that after a power-up the contents of the mapping registers are not defined.

There are 32 mapping registers which may be written and read. These registers are 21 bits wide, and require two Unibus transactions for each read or write; 64 addresses on the I Page (17 770 200 - 17 770 376) are thus allotted to them. The contents of the mapping registers are added to the Unibus address during the relocation process. It should be noted that the last mapping register (addresses 17 770 374 and 17 770 376) can be read and written, but cannot be used to map Unibus addresses because it would be used by addresses in the range of 17 760 000 - 17 777 777; the upper limit jumpers cannot recognize these as valid Cache Unibus addresses. Refer to Chapter 4.

CHAPTER 1
GENERATION OF THE PHYSICAL ADDRESS

Relocation expands the 18-bit Unibus address to the 22-bit Main Memory address. This allows the Unibus to access any location in Main Memory. This relocation, or mapping of addresses, is done by adding the contents of one of the mapping registers to bits (12:01) of the incoming Unibus address.

1.1 CONSTRUCTION OF A PHYSICAL ADDRESS

All mapping registers in the Unibus Map are 21 bits wide. A "22nd" bit, which is not writable and is always read as a zero, acts as the lowest order bit for each register. Each register specifies the 21-bit Physical Address (PA) of a 4K page residing on any word boundary in memory. The reason for using word boundaries in the mapping registers is that the mapping box does not know if a byte operation is being executed, and if so, what byte is required.

Refer to Figure 1-1. Bits (17:13) of the 18-bit Unibus address select which register a device is using. The remaining bits (12:00) of the Unibus address act as an offset into the page to which the mapping register is pointing.

When an address is taken off the Unibus, the mapping register is automatically selected and the contents read out. That 21-bit address is added to the 13-bit offset in the Unibus address to form the PA. This mapping function is very similar to that performed by Memory Management.

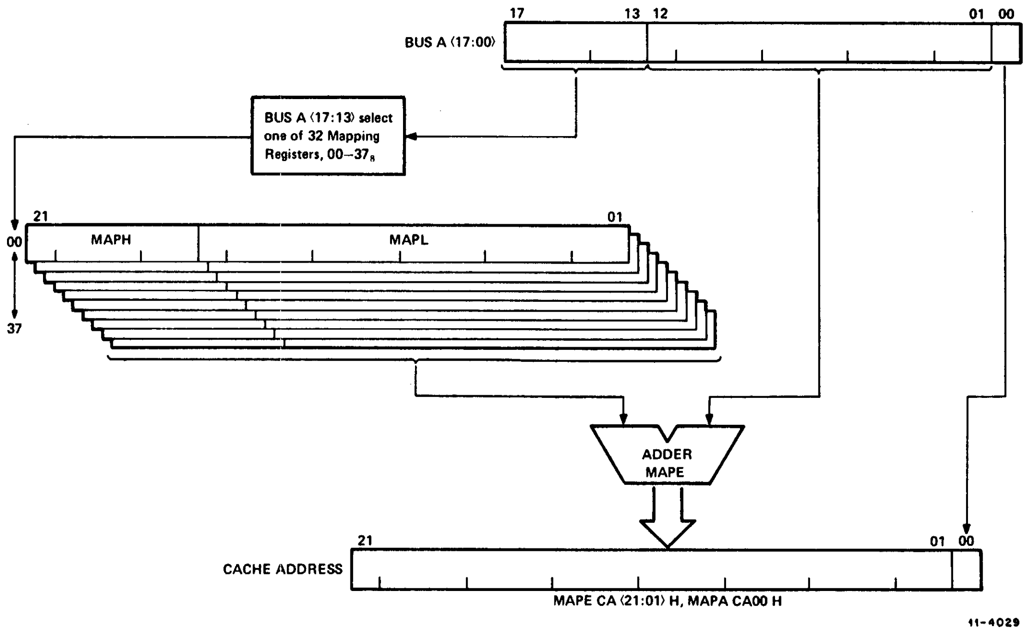

17
13 12
01 00
BUS A (17:00)
BUS A (17:13) select one of 32 Mapping Registers, 00–37₈
21
01
00
MAPH
MAPL
37
ADDER
MAPE
21
01 00
CACHE ADDRESS
MAPE CA (21:01) H, MAPA CA00 H
11-4029

Figure 1-1 Construction of the PA

The program controls this process both by selecting the contents of the mapping registers and by its ability to enable and disable the Unibus Map relocation function.

The Unibus address lines, BUS A(17:00) L are received by the Map. The output of their bus receivers is labeled MAPA ADRS(17:01) H and MAPA CA00 H. Address bit 0 is always transmitted, unmodified, to the Cache, since the Map ignores byte instructions.

The address used by the Cache during a data transaction consists of MAPA CA00 H, the output of the Map receiver for BUS A00 L, and of MAPE CA(21:01) H. These bits are the output of a 21-bit adder which is enabled when MAPD ENAB MAP is asserted, and disabled when ENAB MAP is negated. The state of this signal reflects that of SCCL ENAB MAP H (bit 5 of Memory Management Register 3).

When the adder is disabled, its output is the same as the incoming Unibus address, with bits 18 - 21 equal to 0.

Refer to Figure 1-1. When the adder is enabled, bits 17 - 13 of the Unibus address-select one of the mapping registers [MAPC+D RA(21:01) H]. The contents of this register are summed with bits 12 to 1 of the Unibus address [MAPA ADRS(12:01) H] to generate the Cache address [MAPE CA(12:01) H].

The mapping registers consist of the 12 3101A 16-word by 4-bit scratch pad memories, shown on drawings MAPC and MAPD.

1.2 REGISTER SELECTION

During a Cache Unibus cycle, MAPB REG OP L is high (i.e., the address does not point to a map register). MAPA ADRS17 H then causes either MAPC EN LO REG L or EN HI REG L to be asserted. These signals enable, respectively, the registers on MAPC (addresses 17 770 200 - 17 770 276) or those on MAPD (addresses 17 770 300 - 17 770 376) through their CS inputs. One of 16 registers is selected by MAPA ADRS(16:13) H via the multiplexer on MAPC. MAPC INDA(4:1) H address the mapping register that is being selected.

1.3 ADDER

The Adder consists of the five 74S181 ALU ICs, and the full adder circuit for bit 21 shown on MAPE.

When MAPD ENAB MAP H is negated, the incoming Unibus address is transmitted, unmodified, through the Adder.

When MAPD ENAB MAP H is asserted, the Adder is enabled. In this case, bits 01 - 12 of the address are added to bits 01 - 21 of the selected mapping register (adder function A plus B). The output of the Adder then goes to the Cache as the PA.

1.4 ADDRESSING LIMITS

Refer to schematic MAPF. There are 31 mapping registers which can be accessed by the Unibus for relocation. The actual number is determined by two sets of five jumpers which set the upper and lower address limits to which the Unibus Map will respond. The jumpers for the lower limit can be cut so that the Map will start to respond at Unibus address 0K, 4K, up to 124K on 4K boundaries. Similarly, the jumpers for the upper limit can be set so that the Map will stop responding at Unibus address 124K, 120K, and down to 0K on 4K boundaries. The Map will not respond to the uppermost 4K of Unibus address space. The maximum range of Unibus addresses that the Map can accept is 000 000 - 757 777.

Bits(17:13) of an incoming Unibus address are checked against the jumpers to ensure that the address lies inside these limits. If the address is greater than or equal to the upper limit, or less than the lower limit, it is assumed that some other device is being addressed and no request is made to the Cache. The Unibus Map can be bypassed altogether by cutting both sets of jumpers to all zeroes. This would mean that Main Memory cannot be accessed from the Unibus.

CHAPTER 2
UNIBUS/CACHE INTERFACE

This chapter describes the Unibus/Cache interface, excluding address relocation, which is explained in Chapter 1.

Figure 2-1 outlines this interface function of the Map, and Figure 2-2 is a functional block diagram. DATA exchanged between the Unibus [BUS D(15:00)] and the Cache is buffered by the Unibus Map and transmitted, without modification, in both directions. The Unibus Control bits (BUS C0 and C1) are received by the Unibus Map and transmitted directly to the Cache. BUS MSYN is sent to the Cache as MAPF UB REQUEST (1) L if the Unibus address [BUS A(17:01) L] is recognized as a valid Cache data or register address. A parity error in the Cache causes BUS PB L to be asserted by the Unibus Map. BUS SSYN L is asserted by the Unibus Map when it is informed by the Cache that the data cycle is finished (CCBC UB DONE H).

The Unibus address is decoded by the Unibus Map. If it is a Cache register address (17 777 740 – 17 777 752), MAPB CACHE REG L is sent to the Cache: this signal, in addition to Unibus address bits MAPA ADRS(03:01) H allows the Cache to select the register required for the current data transaction.

If the address is a valid Cache address (as determined by the limit jumpers), it is either sent to the Cache unmodified (if the Map is not enabled), or relocated (if the Map is enabled). The Map is enabled if bit 5 of Memory Management Register 3 is set (Unibus address 17 772 516). Figure 2-3 shows the signals exchanged between the Unibus, the Map, and the Cache.

The Map responds as a slave to the two major types of Unibus transactions: DATI (or DATIP) which requires a read from memory, and DATO (or DATOB), which requires a write into memory. The Map does not distinguish between DATI (data-in) and DATIP (data-in, pause), nor between DATO (data-out) and DATOB (data-out, byte): it transmits Unibus control bit C0, which distinguishes DATI from DATIP and DATO from DATOB, to the Cache.

2.1 UNIBUS DATA CYCLE

The Unibus address lines, BUS A(17:00) L are received by the Map. The output of their bus receivers is labeled MAPA ADRS(17:01) H and MAPA CA00 H. Address bit 0 is always transmitted, unmodified, to the Cache, since the Map ignores byte instructions.

MAPA ADRS(17:02) are decoded and MAPB CACHE REG is asserted if a Cache register address is sensed [(17) 777 740 – (17) 777 752]. This signal and MAPA ADRS(03:01) H constitute a Cache register address.

MAPA ADRS(17:13) H are compared with the upper and lower limit jumpers shown on schematic MAPF. If the address falls within the limits set by the jumpers, or if a Cache register address has been decoded (MAPB CACHE REG L), MAPF CACHE BUS ADRS L is asserted.

Refer to Figure 2-4. Upon receipt of the assertion of BUS MSYN L, the flip-flop MAPF UB REQUEST (1) L is set; its output starts a Cache read or write sequence. The UB REQUEST flip-flop is reset by CCBC UB ACKN L, which is asserted by the Cache when the Unibus memory cycle is initiated. MAPJ ENBUS H is asserted at the same time as UB REQUEST H. It gates SSYN and the data onto the Unibus lines.

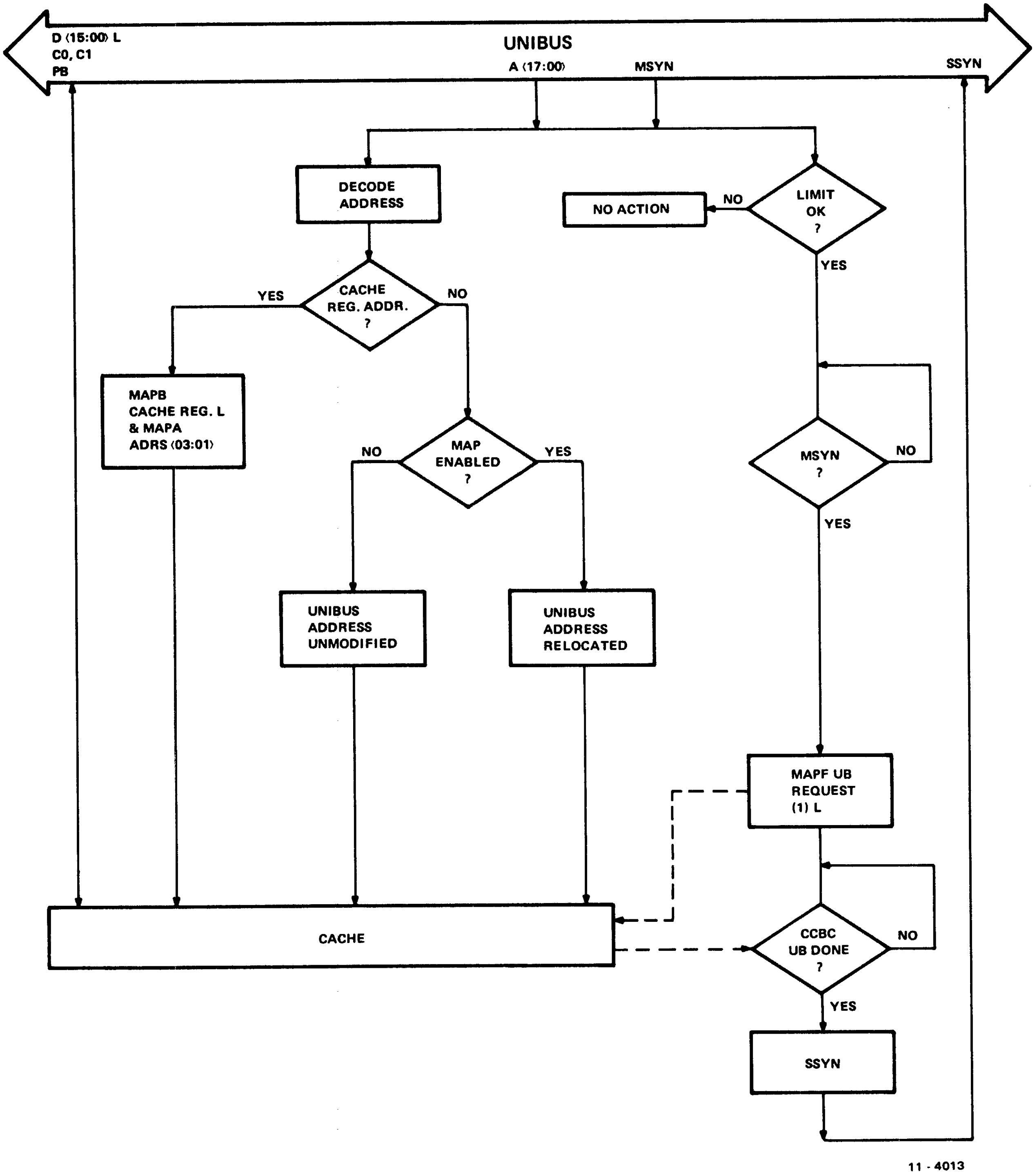


Figure 2-1 Unibus Map Flowchart

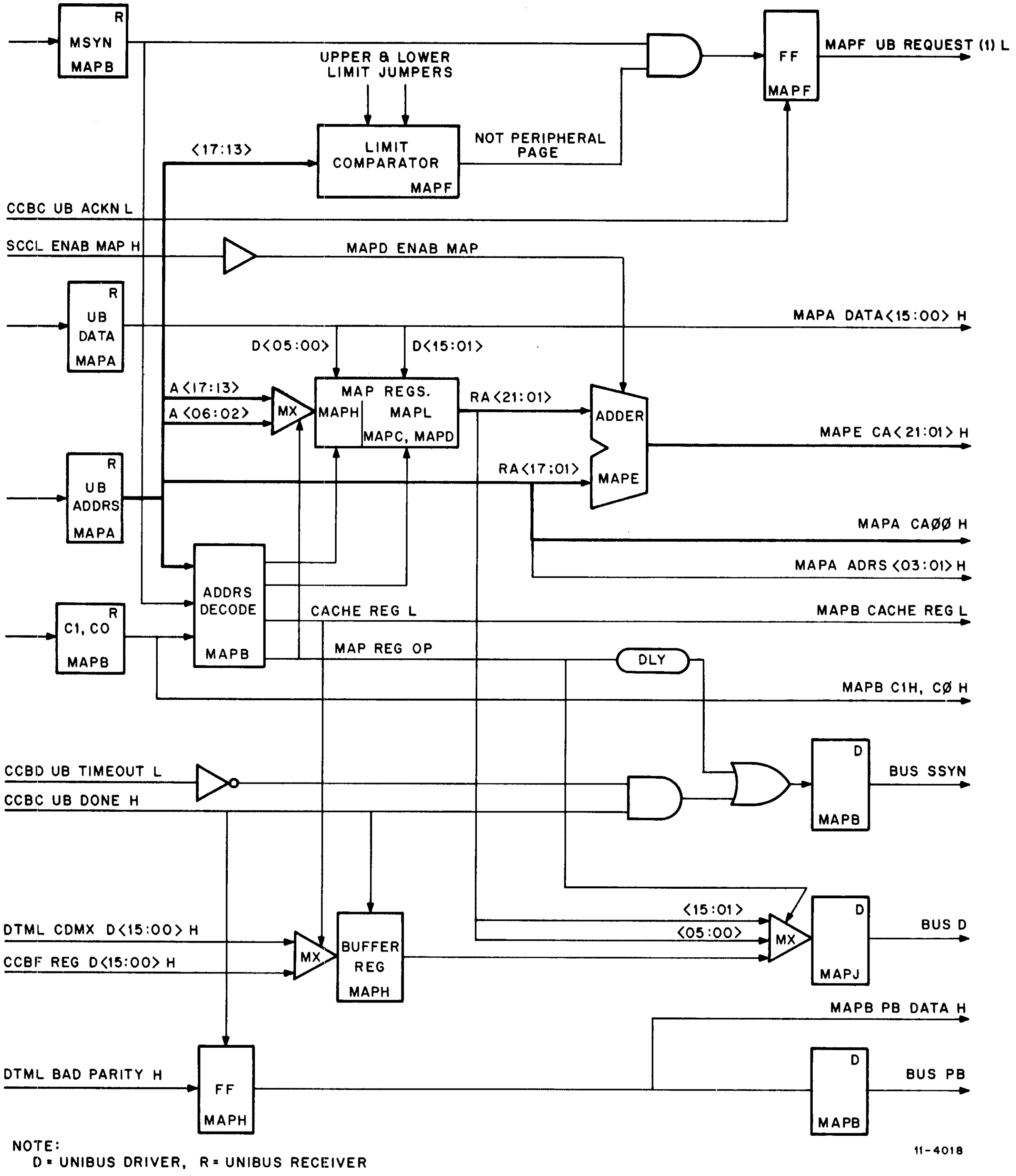


Figure 2-2 Unibus Map Block Diagram

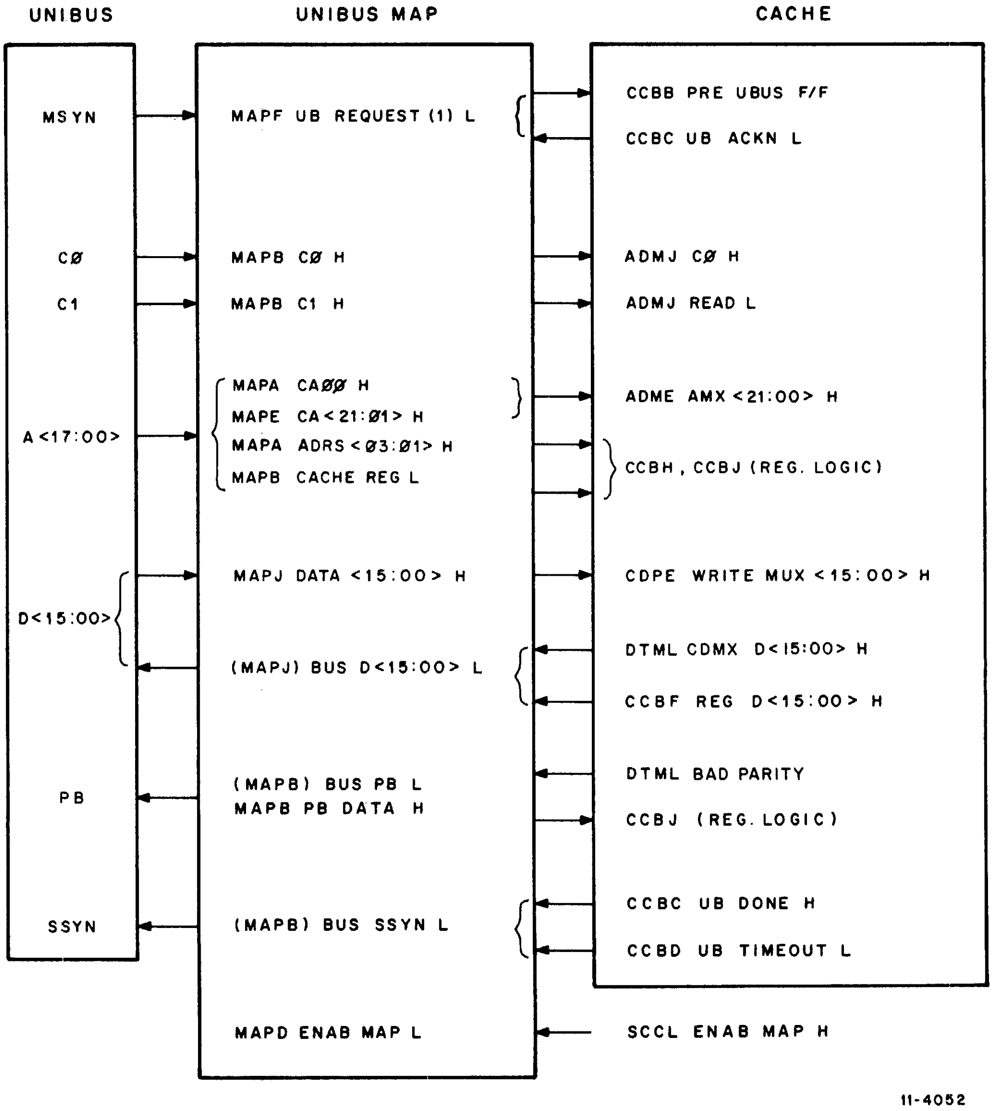


Figure 2-3 Unibus Map Interface

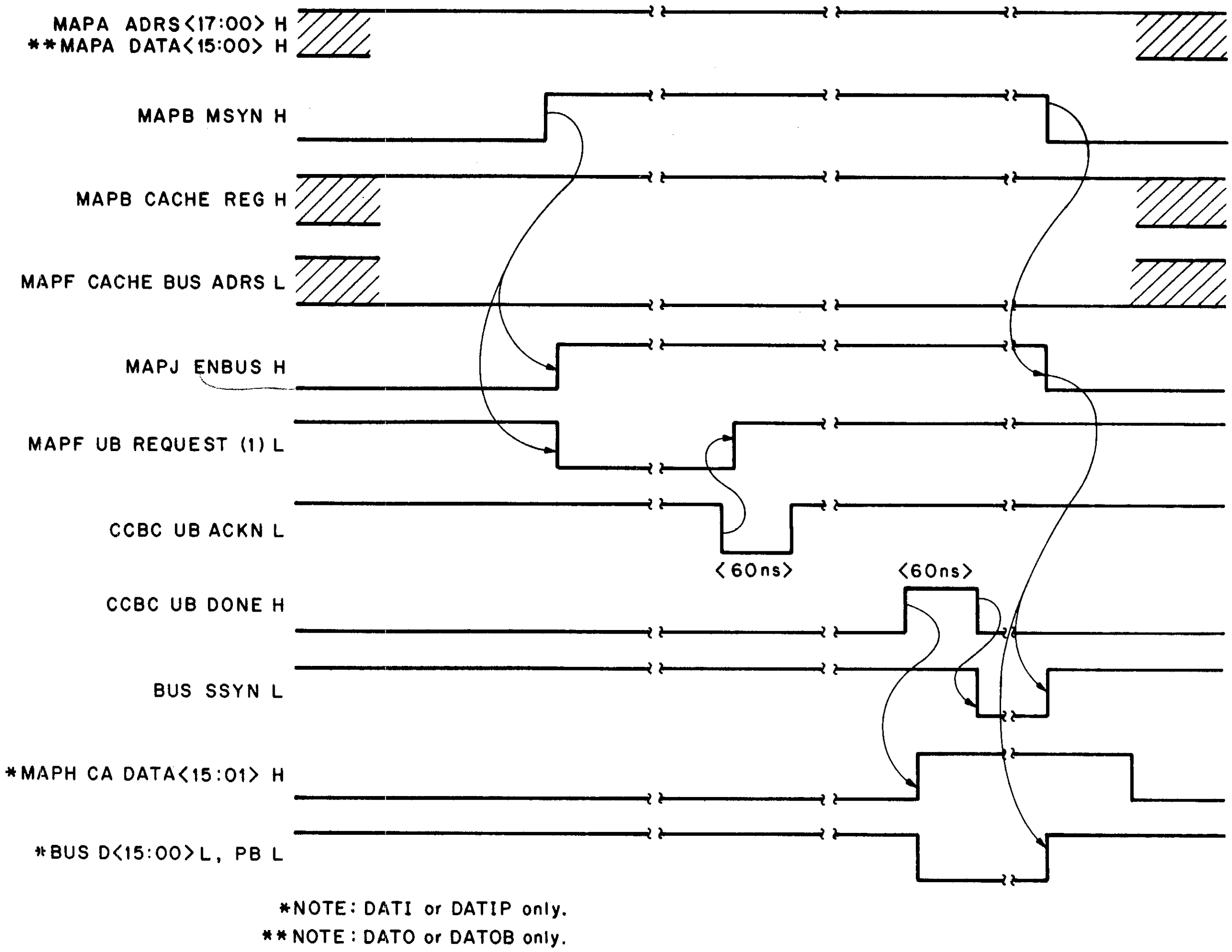


Figure 2-4 Cache/Unibus Transactions

Along with the Control bits C1 and C0, the Cache receives the address MAPE CA(21:01) H and MAPA CA00 H for memory references, or MAPA ADRS(03:01) H and MAPB CACHE REG L for a Cache register reference. When UB REQUEST is received, it executes the write (DATO/B) or read (DATI/P) operation required of it.

2.2 DATO OR DATOB

In the case of a data-out, the Cache accepts the data, MAPA DATA(15:07) H. It then asserts CCBC UB DONE H.

2.3 DATI OR DATIP

If the transaction is a data-in, the Cache puts the requested data on DTML CDMX D(15:00) H, in the case of a memory reference; if tne operation refers to a Cache register, the data is transmitted on CCBF REG D(15:00) H. One of these two sets of data is selected by MAPB CACHE REG L via the multiplexer shown on drawing MAPH.

The output of this multiplexer is clocked into the MAPH CA DATA(15:00) (1) H flip-flops by the rising edge of CCBC UB DONE H. (UB DONE is asserted by the Cache when its data operation is completed.) MAPH CA DATA is then multiplexed with the map register data (MAPJ); since MAPB REG OP H is low (not a map register operation), the Cache data is the input to the Unibus data drivers [BUS D(15:00) L], which are enabled by MAPJ ENBUS H.

2.4 END OF DATA CYCLE

The falling edge of CCBC UB DONE H, which occurs 60 ns after its rising edge, sets a flip-flop on MAPB which in turn causes BUS SSYN L to be asserted. When the negation of MSYN is received at the Map, MAPJ ENBUS H is negated. This causes SSYN and, in the case of a DATI or DATIP, the data and PB, to be removed from the Unibus.

2.5 PARITY ERROR

BUS PB L is asserted when a parity error is detected by the Cache. DTML BAD PARITY H is clocked into a latch on MAPH at the same time as the Cache data. The output of this latch is MAPH PAR ERR (1) H; it is gated with CI and MAPF PAR ADRS OK H to generate MAPB PB DATA H, which is input to the Cache error registers. MAPB PB DATA H is also ANDed with ENBUS to generate BUS PB L. MAPF PAR ADRS OK H, when asserted, signifies that the address of the current transaction lies within the limits of the upper and lower limit jumpers.

2.6 CACHE TIMEOUT

CCBD UB TIMEOUT L is asserted by the Cache when a timeout has occurred on the Main Memory Bus during a Unibus transaction. When asserted, it sets a flip-flop on MAPB which prevents the assertion of BUS SSYN L.

CHAPTER 3
READING AND WRITING THE MAPPING REGISTERS

The mapping registers are loaded and read by the program via the Unibus. These registers are 21 bits wide; two Unibus cycles are required to read them or to write into them. There are 32 mapping registers which require the 64 I/O Page addresses in the range of 770200 - 770376. Each of the registers consists of two parts (for the purposes of reading and writing): a high word, MAPH (bits 21 - 16) and a low word, MAPL (bits 15 - 01). Bit 0 does not exist, since the Map ignores byte operations.

3.1 READING AND WRITING MAPPING REGISTERS

Refer to Figure 3-1. The Unibus Map responds to 64 Unibus Addresses [(17) 770 200 - (17) 770 376]. This allows reading and writing of the mapping registers. Once the Map has recognized one of these addresses, it uses bits (06:02) to select the correct register [as opposed to bits (17:13) for a mapping operation]. Sixty-four addresses are needed due to the 22-bit register width.

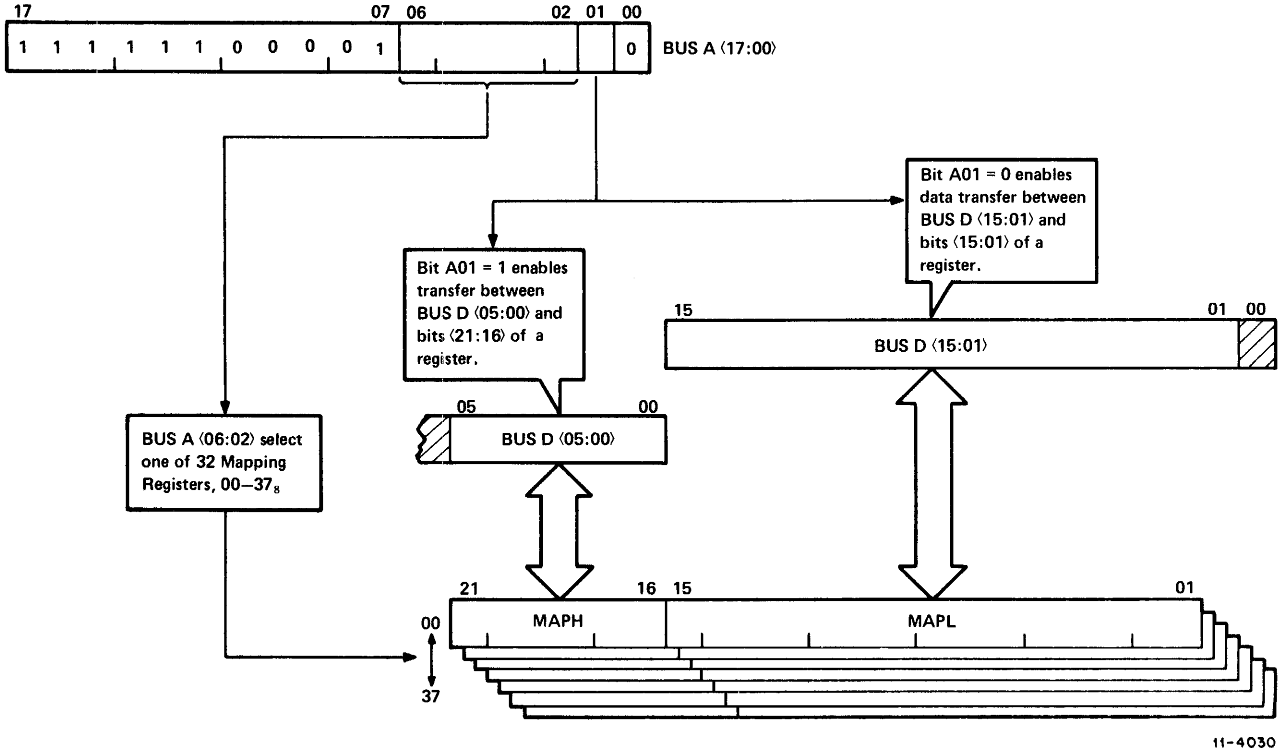

17
07 06
02 01 00
1 1 1 1 1 1 0 0 0 0 1
0
BUS A ⟨17:00⟩
Bit A01 = 0 enables data transfer between BUS D ⟨15:01⟩ and bits ⟨15:01⟩ of a register.
Bit A01 = 1 enables transfer between BUS D ⟨05:00⟩ and bits ⟨21:16⟩ of a register.
15
01 00
BUS D ⟨15:01⟩
05
00
BUS D ⟨05:00⟩
BUS A ⟨06:02⟩ select one of 32 Mapping Registers, 00–37₈
21
16 15
01
00
MAPH
MAPL
37
11-4030

Figure 3-1 Addressing of UB Map Register

Also as a result of this, two Unibus cycles are required to complete a read or write operation to a mapping register. The bit assignment in the registers is divided so that Unibus address (17) 770 XXX will access bits (15:01) of the register and address (17) 770 XXX+2 will access bits (21:16).

3.2 REGISTER SELECTION

MAPB REG OP is asserted when an Unibus address in the range of 770 200 - 770 376 is decoded.

Refer to Figure 3-2. MAPB REG OP H is gated with MAPA ADRS06 H to select either registers 00 - 17_8 on MAPC (MAPC EN LO REG L), or registers 20_8 - 37_8 on MAPD (MAPC EN HI REG L) by enabling the 3101s via the enable (CS) input.

MAPB REG OP L gates MAPA ADRS(05:02) H to MAPC INDA(4:1), which in turn select one of 16 registers (either one of 00–17_8 or 20–37_8, depending on which set of CS inputs is low).

MAPB REG OP L gated with MSYN causes MAPJ ENBUS H to be asserted. ENBUS gates SSYN and, during a DATI, the register data onto the Unibus.

3.3 DATO

A DATO is a write to a register. MAPB C1 H is asserted and, when MSYN is received, either MAPB WRITE HI WORD L or WRITE LO WORD L is asserted, depending upon the state of MAPA ADRS01. WRITE HI WORD gates bits 05 - 00 of Unibus data into bits 21 - 16 of the selected register; WRITE LO WORD gates bits 15 - 01 into bits 15 - 01 of the register.

The receipt of MSYN also starts a 70-ns delay, which allows for the write propagation time of the 3101As. At the end of the delay, MAPB REG SSYN L turns off the write pulse and causes BUS SSYN L to be asserted. When BUS MSYN is negated, SSYN is negated.

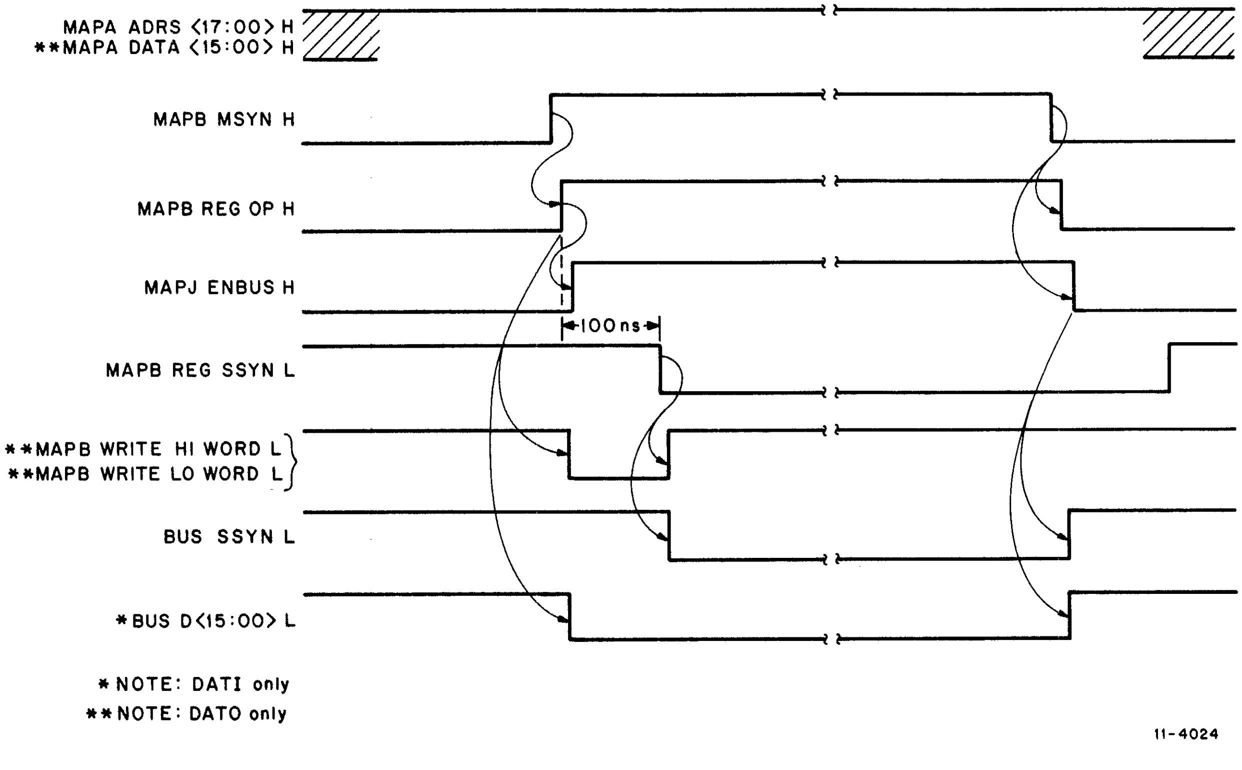


Figure 3-2 UB Map Register Read/Write

3.4 DATI

When a register has been selected, the output of the selected register [MAPC+D RA(21:01) H] is read and input to the data multiplexer shown on drawing MAPJ. Since MAPB REG OP H is high, MAPA ADRS01 selects (via the multiplexer) either the low word (MAPL) or the high word (MAPH) of the register that is being addressed, as shown in Table 3-1. When the MAPH part of a register is selected [RA(21:16)] the A inputs to the 74S157 multiplexers are selected. This is the Cache data input [MAPH CA DATA(15:06) H], but it is 0, since the flip-flops on drawing MAPH were cleared by the negation of MSYN on the previous reference. Thus, bits 15 – 06 of BUS D are all 0, and BUS D(05:00) contain the high order bits (21:16) of the selected register.

The multiplexer is enabled if the operation is a DATI (MAPB C1 H is low). The 8881 bus drivers [BUS D(15:00) L] are enabled by MAPJ ENBUS H, thus putting the contents of the selected register on the Unibus D lines.

When MSYN is received, the 70-ns delay is initiated to allow for the access propagation times of the 3101As. When MAPB REG SSYN L is asserted after the delay, BUS SSYN L is asserted on the Unibus. SSYN is negated upon receipt of the negation of MSYN.

3.5 REGISTER ACCESS

Table 3-2 shows the correspondence between the Unibus addresses that select each mapping register and the two addresses used for reading or writing the same register.

Note that register 37 is selected by Unibus addresses (17) 760 000 – (17) 777 777. Since these addresses are higher than the maximum allowed by the upper limit jumpers, register 37 cannot be used as a mapping register. It can, however be read and written into by using addresses (17) 770 374 and (17) 770 376.

Table 3-1
Unibus Data Selection

MAPB REG OP H	MAPA ADRS01	Bus D(15:06)	Bus D(05:01)	Bus D00
	L H			
H	L H	0	MAPC+D RA(21:17)	MAPC+D RA16
H	H L	MAPC+D RA(15:06)	MAPC+D RA(15:01)	0

Table 3-2
Access to Unibus Map Registers

Register No.	Unibus Address Read or Write		Unibus Address for Memory Reference
	MAPL	MAPH	
0	17 770 200,	02	17 000 000 – 17 017 777
1	17 770 204,	06	17 020 000 – 17 037 777
2	17 770 210,	12	17 040 000 – 17 057 777
3	17 770 214,	16	17 060 000 – 17 077 777
4	17 770 220,	22	17 100 000 – 17 117 777
5	17 770 224,	26	17 120 000 – 17 137 777
6	17 770 230,	32	17 140 000 – 17 157 777
7	17 770 234,	36	17 160 000 – 17 177 777
10	17 770 240,	42	17 200 000 – 17 217 777
11	17 770 244,	46	17 220 000 – 17 237 777
12	17 770 250,	52	17 240 000 – 17 257 777
13	17 770 254,	56	17 260 000 – 17 277 777
14	17 770 260,	62	17 300 000 – 17 317 777
15	17 770 264,	66	17 320 000 – 17 337 777
16	17 770 270,	72	17 340 000 – 17 357 777
17	17 770 274,	76	17 360 000 – 17 377 777
20	17 770 300,	02	17 400 000 – 17 417 777
21	17 770 304,	06	17 420 000 – 17 437 777
22	17 770 310,	12	17 440 000 – 17 457 777
23	17 770 314,	16	17 460 000 – 17 477 777
24	17 770 320,	22	17 500 000 – 17 517 777
25	17 770 324,	26	17 520 000 – 17 537 777
26	17 770 330,	32	17 540 000 – 17 557 777
27	17 770 334,	36	17 560 000 – 17 577 777
30	17 770 340,	42	17 600 000 – 17 617 777
31	17 770 344,	46	17 620 000 – 17 637 777
32	17 770 350,	52	17 640 000 – 17 657 777
33	17 770 354,	56	17 660 000 – 17 677 777
34	17 770 360,	62	17 700 000 – 17 717 777
35	17 770 364,	66	17 720 000 – 17 737 777
36	17 770 370,	72	17 740 000 – 17 757 777
*37	17 770 374,	76	17 760 000 – 17 777 777

*Note: Can be read or written into, but not used for mapping.

SECTION VI

CACHE

Unless otherwise indicated, references within this section pertain to this section only.

SECTION VI CACHE
CONTENTS

Page

CHAPTER 1 CACHE CONCEPTS

CHAPTER 2 PDP-11/70 CACHE

CHAPTER 3 THEORY OF OPERATION

CONTENTS (Cont)

ILLUSTRATIONS

TABLES

TABLES (Cont)

CHAPTER 1
CACHE CONCEPTS

1.1 SCOPE

This chapter explains the purpose of cache memory systems and describes various methods used to implement such systems. Parameters and strategies involved in cache memory design are introduced, described, and analyzed in order to facilitate the reader's understanding of the specific Cache implemented in the PDP-11/70 system.

1.2 OVERALL ORGANIZATION OF A CACHE MEMORY SYSTEM

The cache memory system is intended to simulate a system having a large amount of fast memory. To do this, the cache system relies on a small amount of very fast memory (the cache), a large amount of slower memory (the Main Memory), and the statistics of program behavior.

The basic idea is to store some data in the fast memory and some in the slow memory. If it can somehow be arranged that data is in the fast memory when the processor needs it, the program will execute quickly, slowing down only occasionally for Main Memory operations. Conventional mixed MOS-Core systems attempt to achieve this goal by having the programmer guess beforehand which sections of his program should go in each memory. This is often awkward, and usually only moderately successful. The cache memory system tries to achieve the same goal by automatically, dynamically shuffling data between the two memory types in a way which gives a high probability that useful data will be in the fast memory. All of the following discussions of cache organizations and strategies are intended to show implementable methods of shuffling data so that the data most likely to be needed next will be in the fast memory instead of the slower Main Memory.

Figure 1-1 illustrates the relationship of a cache to the processor and Main Memory.

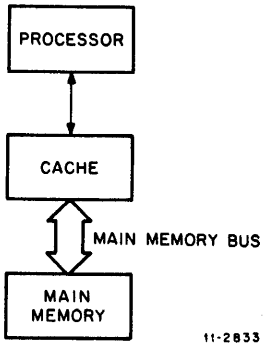


Figure 1-1 Relationship of Cache to Processor and Main Memory

1.3 PROGRAM LOCALITY

A cache memory works because it can usually predict successfully which words a program will require soon. If programs used words completely at random from all of memory, it would be impossible to predict which words would most likely be needed next. Under these circumstances, a cache memory system could perform no better than a conventional mixed memory system with a small amount of bipolar memory.

Fortunately, programs do not generate random addresses. Instead, programs have a tendency to make most accesses in the neighborhood of locations accessed in the recent past. This is the basis of the principle of program locality. The fact that programs display this type of behavior makes cache memory systems possible.

An understanding of why the principle of program locality is true can be obtained by examining the small scale behavior of typical program data structures. Code execution itself generally proceeds in straight lines or small loops; the next few accesses are most likely to be within a few words ahead or behind. Stacks grow and shrink from one end, with the next few accesses near the current top. Character strings and vectors are often scanned through sequentially.

The principle of program locality is a statement of how most programs tend to behave, not a law which all programs always obey. Jumps in code sequences, seemingly random access of symbol tables by assemblers, and context switching between programs are examples of behavior which can adversely affect the locality of addresses generated by a processor. The process of guessing which words a program will reference next can never be completely successful. The percentage of correct guesses is a statistical measure affected by the size and organization of the cache, the algorithms it uses, and the behavior of the program driving it.

1.4 BLOCK FETCH

The principle of program locality states that for the cache to have the best chance of having the word the program needs next, the cache should have words near those recently used. The basic method of accomplishing this is the block fetch. When the cache controller finds it necessary to move a word of data from slow memory to fast memory because the data was not in the fast memory when needed, the controller will move not just the word required, but a block of several adjacent words at once. Typically, the block will contain one (degenerate case), two, four, or eight words starting on an even block boundary.

The block fetch can provide either look-behind, look-ahead, or both, depending on the position of the originally requested word within the block. Since many important generated address sequences (e.g., most code) tend to move in increasing order, the originally requested word is usually the first in the block, so the block fetch generally provides look-ahead.

The block size is one of the most important parameters in the design of a cache memory system. If the block size is too small, the system will have insufficient look-ahead and performance will suffer slightly, particularly for programs which do not contain many loops. Also, as will be discussed later, small block sizes require the system to store more addresses than large blocks, for the same total memory size.

If the block is too large, there may not be room for enough blocks in the cache to provide for adequate look-behind. Large blocks also tend to mean more memories operating in parallel within the slow memory, and therefore wider buses between slow and fast memory, resulting in increased cost. As the block gets larger, each additional word in the block is less likely to be useful, since it is further from the originally requested word and less likely to be needed soon by the program. It has been found empirically that while a block size of two words increases memory system performance dramatically, further increases in block size produce much smaller improvements which are seldom worth implementing.

1.5 FULLY ASSOCIATIVE CACHE

If a cache memory system was designed so that the fast memory held one contiguous block of 1000 words, it would fail miserably. Most programs make reference to code segments, subroutines, stacks, lists, and buffers located in scattered parts of the whole address space. Ideally, a 1000-word cache would hold the 1000 words the controller estimated as most likely to be needed, no matter how scattered these words were throughout the address space of Main Memory.

Since there would be no relation of all the addresses of these thousand words to each other or to any single register or mapping function, each of the 1000 data words in the fast memory would have to carry its address with it. Then, when the processor requested a word from memory, the cache would simply compare (associate) the address from the processor with each of the thousand addresses of words in the fast memory. If a match were found, the data for that address would be sent to the processor. This is the principle of an associative memory (Figure 1-2).

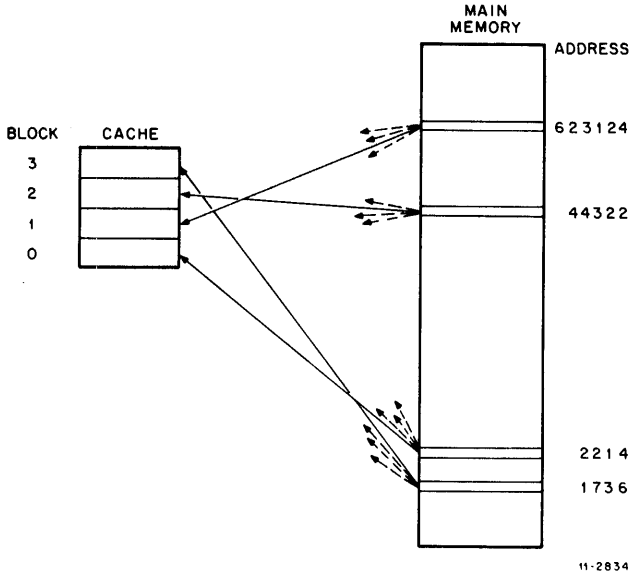


Figure 1-2 Fully Associative Cache Memory System

This system, called fully associative because the incoming address must be compared (associated) with all the stored addresses, gives the cache controller maximum flexibility in deciding which words it wants in fast memory, i.e., any words at all until the memory is full. Unfortunately, 1000 address comparisons would be unacceptably slow and/or expensive. One of the basic issues of cache organization is how to provide minimum restrictions on what groups of words may be present in fast memory, while limiting the number of address comparisons required.

1.6 DIRECT MAPPING CACHE

At the opposite extreme from the fully associative cache is the direct mapping cache. Instead of one address comparison on every block, the direct mapping cache requires only one address comparison.

The many address comparisons of the fully associative cache are necessary because any block from Main Memory can be placed in any block of fast memory. Thus, every block of fast memory must be checked to see if it has each requested address. The direct mapping cache allows each block from Main Memory only one possible location in fast memory (Figure 1-3). Consider each incoming address as being made up of three parts. The first part starts at bit 0 and contains enough bits to specify which byte out of a block is being requested. The next field, called the index field, starts where the first field leaves off and contains enough bits to specify any block in fast memory. The third field, called the address field, contains the rest of the bits.

As an example, consider an 18-bit PDP-11 byte address as input to a 256-word, 4 word per block direct mapping cache. (This cache would thus be 4 words wide and 64 blocks deep. Assuming four words per block allows us to break down the address conveniently, using octal notation.) As illustrated in Figure 1-4, the word field in this case comprises bits 2, 1, and 0, where bit 0 indicates the byte, and bits 2 and 1 indicate the word. The index field comprises bits 8 through 3, and indicates the block. The address field comprises bits 17 through 9.

If the processor requests word 274356, the cache controller looks at the address which goes with the information currently in block number 35 in fast

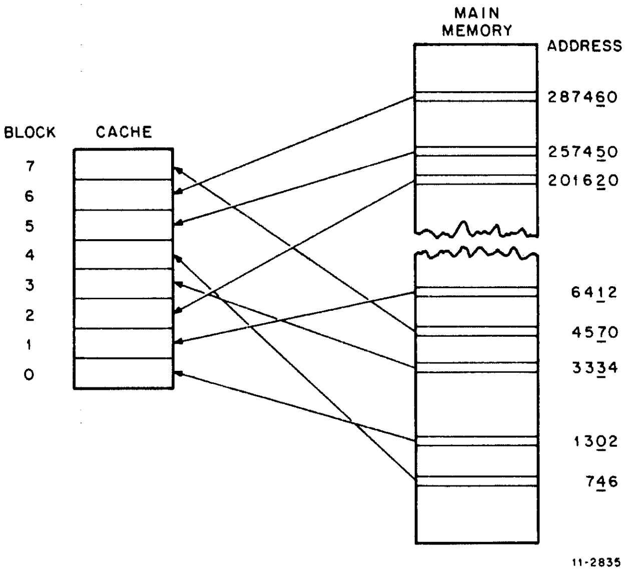


Figure 1-3 Direct Mapping Cache Memory System

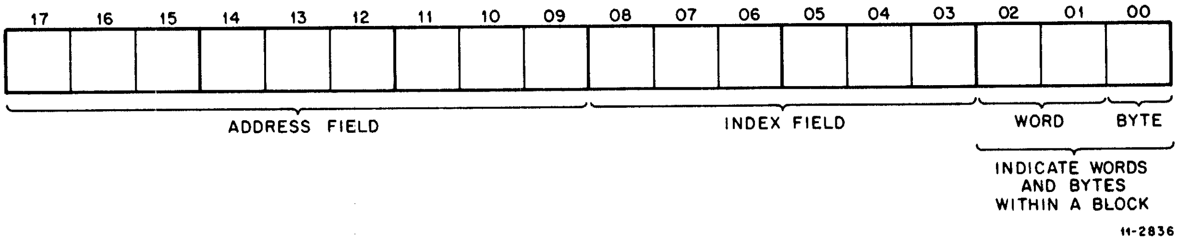


Figure 1-4 18-Bit Byte Address Breakdown (4 Words per Block, 64 Blocks)

memory. If this address field is 274, the controller sends the third word in that block to the processor. If the stored address field is not 274, the controller must fetch block 27435 from Main Memory, transmit the third word in the block to the processor, load the block into block 35 of fast memory, replacing whatever was there previously, and change the address field stored with block 35 to 274.

Any address whose index field is 35 will be loaded into block 35 of fast memory, and therefore this is the only place the cache controller has to look if the processor requests the data from an address whose index field is 35.

Notice also that only the address field of the address need be stored with each block, because only the address field of the address is required for comparison. The index field need not be compared because anything stored in fast memory block 35 has an index field of 35. The word field need not be compared because if the block is there, every word in the block is there.

This is how the direct mapping cache uses inexpensive direct addressing of fast memory to eliminate almost all comparison operations.

Of course there are disadvantages to this simple scheme. If the processor in the example above makes frequent references to both location 274356 and location 6352, there will be frequent references to slow memory, because only one of these locations can be in the cache at one time. Fortunately, this sort of program behavior is infrequent, so that the direct mapping cache, although offering significantly poorer performance than fully associative, is adequate for some applications. Usually the system of choice is a compromise between a direct mapping cache and fully associative cache, called the set associative cache.

1.7 SET ASSOCIATIVE CACHE

The set associative organization is a compromise between the extremes of fully associative and direct mapping. This type of cache has several directly mapped groups (Figure 1-5). For each index position in fast memory there is not one block, but a set of several, one in each group. (The set of blocks corresponding to an index position is called a "set.") A block of data arriving from Main Memory can go into any group at its proper index position.

Since there are several places for data with the same index field in their addresses to be stored, the type of excessive Main Memory traffic possible in a direct mapping organization is less likely to occur. This gives a set associative cache higher performance. In fact, a four-way set associative cache (four groups) will normally perform very nearly as well as a fully associative cache.

The price that is paid for higher performance is some increase in complexity. There are several places in fast memory where any given piece of data can be stored, so the controller must do several compares (i.e., must associate) to determine in which place (if any) the requested data is located. The number of times it must compare (associate) is of course equal to the number of groups, usually two, three, or four. A set associative cache can be classified as an n-way set associative cache, where n is the number of compares performed (i.e., the number of groups).

Another aspect of the increased complexity becomes apparent when a block of fast memory must be overwritten. There are now several locations in fast memory where the new data from Main Memory may be written (one in each group), so the controller must have some means of deciding which block will be overwritten. The decision could be made using any of the following considerations:

Least Recently Used (LRU) - The block least recently used is replaced.

First In-First Out (FIFO) - The block which has been stored the longest time is replaced.

Random - Blocks are replaced in a random manner.

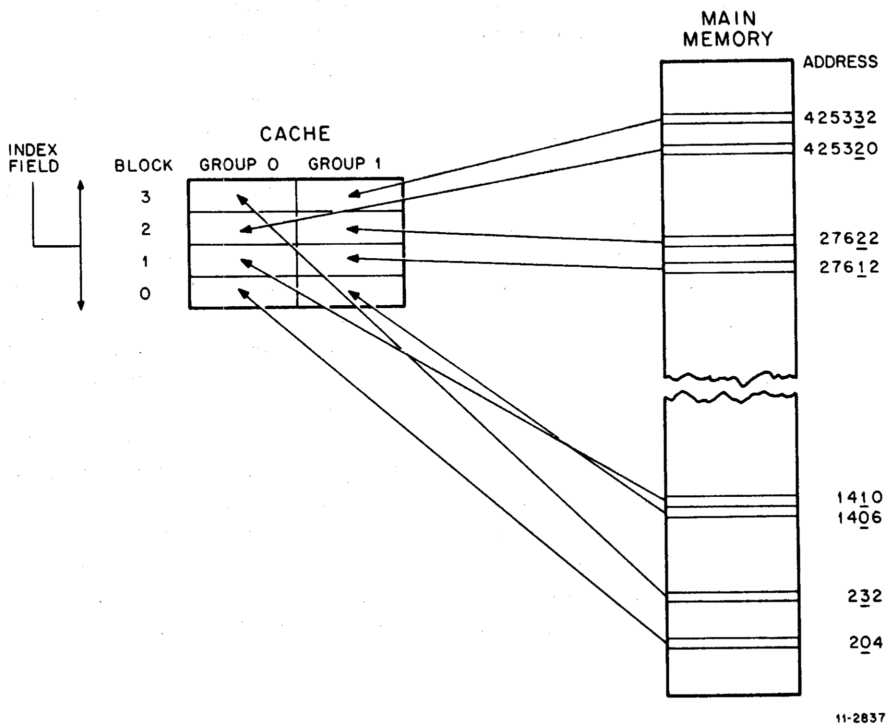


Figure 1-5 Set Associative Cache Memory System (Two-Way)

A replacement strategy based on LRU or FIFO information requires the storage of LRU or FIFO bits, along with the address fields in the address memory, and the logic necessary to generate and decode these bits. The random strategy is far easier and cheaper to implement, yet provides performance only slightly lower than that obtainable by the other strategies.

The extra performance of a set associative cache usually justifies the slightly extra complexity of at least two-way associativity in all but low performance applications.

1.8 WRITE-THROUGH AND WRITE-BACK

Assume that the following sequence of events occur. First, the processor does a read of location 200, resulting in the block containing this address being copied into fast memory. Then the processor writes new data into location 200, updating this location in fast memory. Next the processor does a reference which causes the cache controller to overwrite the block in fast memory containing location 200. If the processor reads location 200 again, the obsolete data in Main Memory will be loaded into fast memory. This is unacceptable, and two methods have been devised to deal with the problem. The methods are called write-through and write-back.

With write-through, whenever a write reference occurs, the data is not only stored in fast memory, but is also immediately copied into Main Memory. This means that the Main Memory always contains a valid copy of all data. If the controller wants to overwrite a block in fast memory, this can be done immediately, without losing any data.

The advantages of write-through are its relative simplicity and the fact that the Main Memory always has correct data. The primary disadvantage is some reduction of speed due to the need to access the slow memory on every write reference. This is offset somewhat by the fact that write references are a small fraction of all references to memory. In addition the cache does not have to wait for the Main Memory to finish before starting the next cycle. Since a reasonable design would only cycle the memory being written into and not all the parallel memories in Main Memory, the system should not even be held up by multiple sequential writes. However, some fraction of the time, the system will have a read miss following a write, or two writes to the same memory stack within Main Memory, and then the system must wait. This causes the system to run slightly slower than first-order estimates would indicate.

The other method of handling the stale data problem in a cache system is called write-back. Under this method, data written by the processor is only stored in the fast memory, leaving the Main Memory unaltered and obsolete. A bit in the address field of the block in fast memory, called the altered bit, is set to indicate that this block contains new information. When the controller wants to overwrite a block of fast memory, the altered bit is inspected first. If this bit is set, the controller must write the block into Main Memory before overwriting it.

The primary advantage of write-back is higher performance. For almost any program, the number of times an altered block must be copied into Main Memory is less than the number of write references, so write-back is noticeably faster than write-through. One disadvantage of write-back is increased complexity. A write-back system must have the ability to regenerate addresses from tags and the extra sequencing logic to do double cycles.

Another disadvantage of write-back is the power fail problem. When power fails, fast memory will be holding the only valid copies of some arbitrary set of locations. If these are not copied into Main Memory, they will be lost. Since there is no way of knowing which locations were lost, the entire memory must be considered volatile. If Main Memory is volatile anyway, there is no problem; otherwise, steps must be taken. One possibility is to require the power fail program to do a sequence of reads calculated to ensure that every block in the cache has been overwritten. A more reliable, but more expensive system would automatically ensure that all altered blocks are copied into Main Memory, after the program halts, but before power disappears.

CHAPTER 2
PDP-11/70 CACHE

2.1 SCOPE

This chapter describes the specific Cache which has been implemented in the PDP-11/70 system. The reader should be familiar with the cache concepts, classifications, and definitions described in the previous chapter.

2.2 PDP-11/70 CACHE

The Cache used in the PDP-11/70 is two-way set associative. It consists of two groups of 256 blocks each. Each block consists of two words; therefore, the total data storage capacity of the fast memory is 1K words. The 11/70 Cache is implemented using a random replacement strategy and write-through.

Since the PDP-11/70 system uses a 22-bit address space, the address is broken down into address field, index field, and word field as illustrated in Figure 2-1 and outlined below.

1. Bits 21–10 comprise the address field used to identify a block of data in fast memory.
2. Bits 9–2 comprise the index field used to designate a set. A set consists of two blocks, one in each group, located at the index position designated by the index field.
3. Bit 1 designates the word (one of two in the block).
4. Bit 0 indicates the byte, as in all PDP-11 addresses.

NOTE

This manual uses the term "address field" to designate that part of an address which is stored in the Address Memory. The term "address tag" designates the tag used to identify data stored in the Cache. The address tag thus consists of an address field, a Valid bit, and two parity bits.

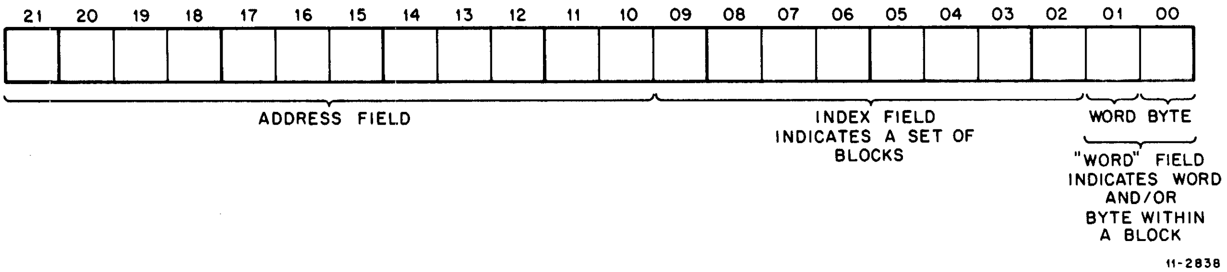


Figure 2-1 22-Bit Byte Address Breakdown (2 Words per Block, 256 Sets of Blocks)

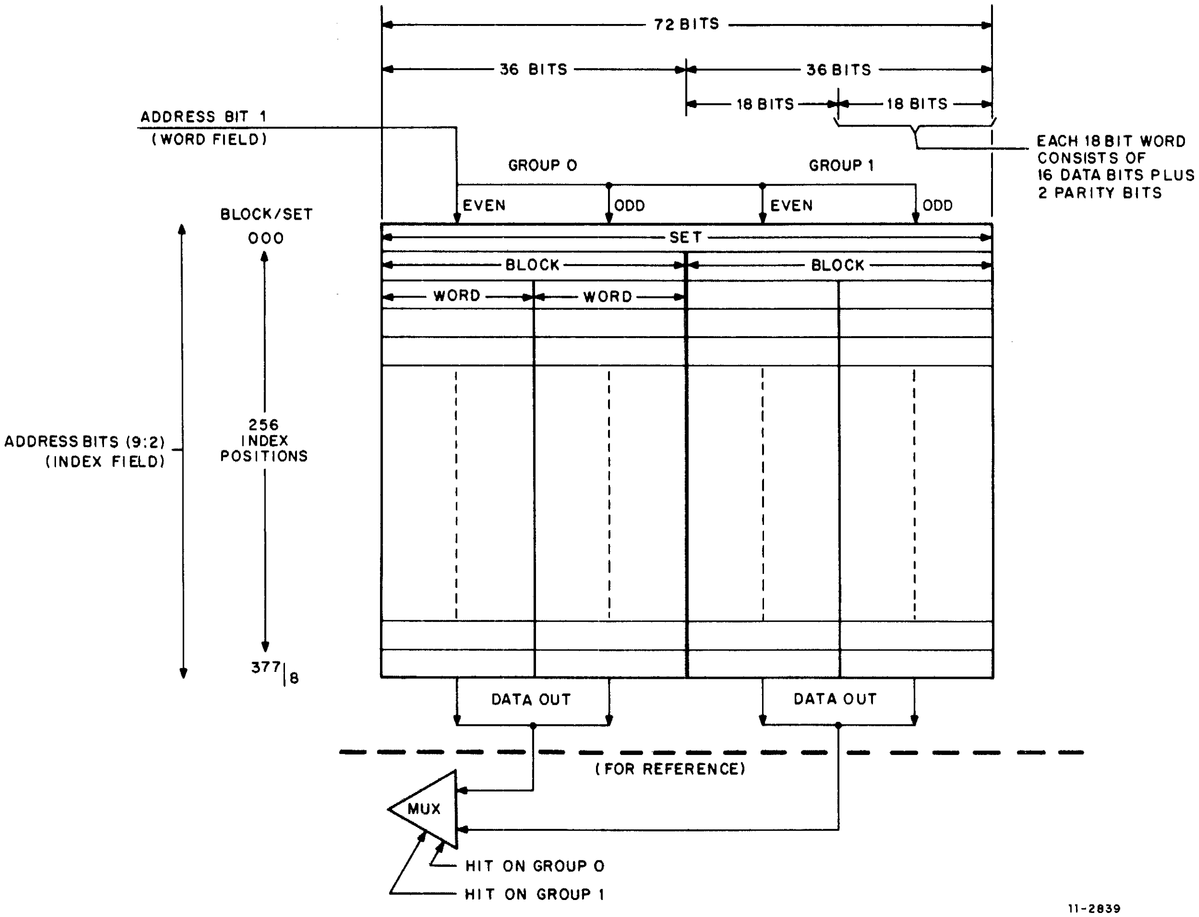


Figure 2-2 Fast Data Memory Organization

2.2.1 Data Memory Organization

Figure 2-2 illustrates the organization of the PDP-11/70 Cache Fast Data Memory (FDM). Note that the FDM consists of 256 sets = 512 blocks = 1024 words, and is subdivided into two equal groups (Group 0 and Group 1). Bits (9:2) of the incoming address index into the FDM and select one of the 256 sets. (A set consists of two blocks, one in each group.) Bit 1 of the incoming address enables either the low (even) word or the high (odd) word within the blocks that comprise the selected set to be gated out of the FDM to a Cache Data Multiplexer. One of these words will be selected if a hit is detected upon address field comparison. The word selected will be from the group upon which the hit occurs.

2.2.2 Address Memory Organization

The organization of the Address Memory, illustrated in Figure 2-3, is determined by the Fast Data Memory organization. The Address Memory is divided into two equal parts: the Tag 0 Address Memory and Tag 1 Address Memory, corresponding to Group 0 and Group 1 of the Fast Data Memory. Since an address tag field must be stored to identify each block in the FDM, 512 locations are required for address tag fields; 256 of these locations are in Tag 0 Address Memory, while the remaining 256 locations are in Tag 1 Address Memory. Each address tag consists of 15 bits. Therefore, the total width of Address Memory is 15 × 2 = 30 bits.

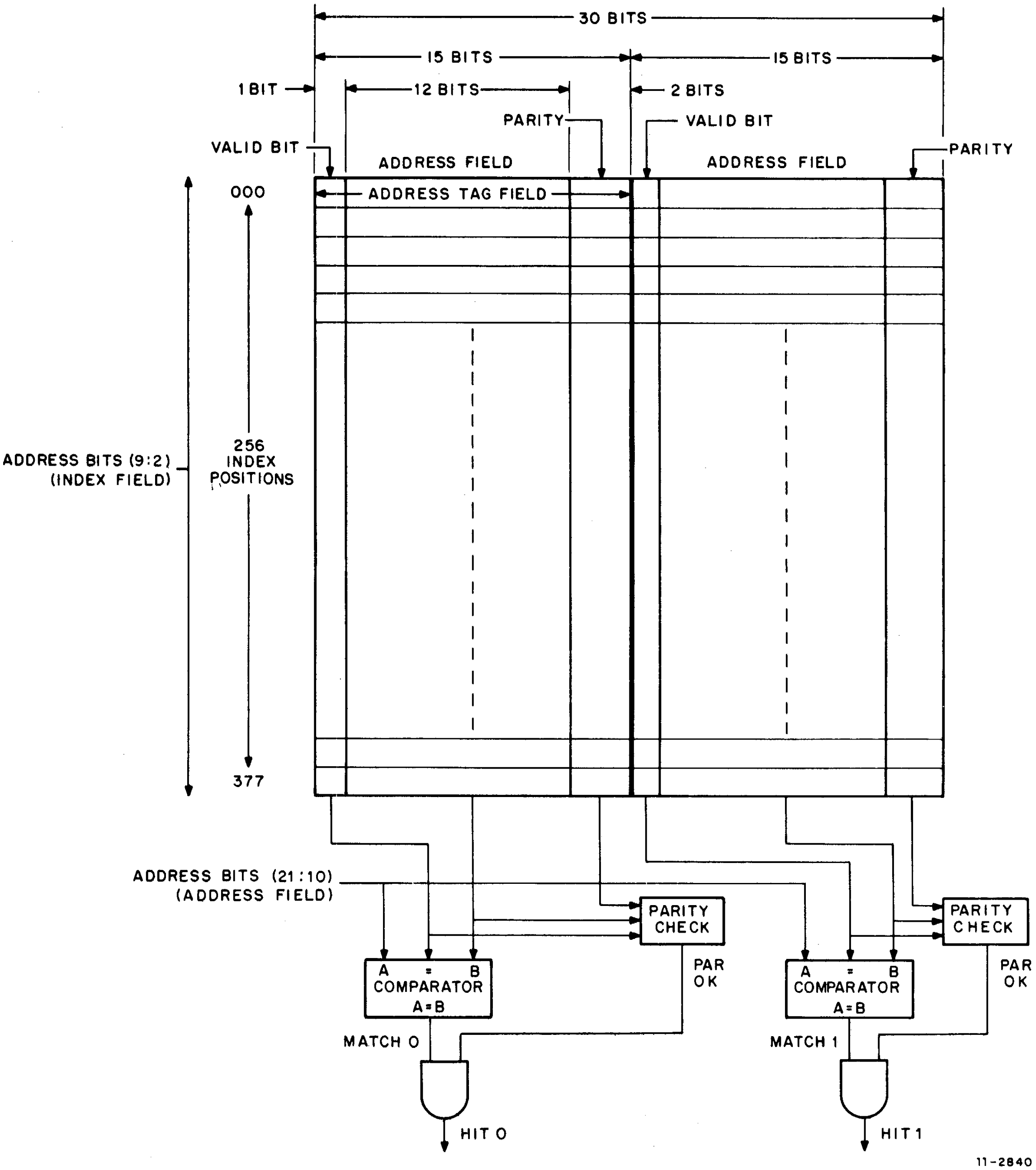


Figure 2-3 Address Memory Organization

The address tag is organized as follows:

1. Twelve bits are required to store the address field.

2. One bit is used to store a Valid bit; this bit indicates whether the address tag (and therefore the FDM data corresponding to the tag) is valid.

3. The remaining two bits are used to store the address tag parity bits which verify that the address tag has been properly loaded into the Address Memory.

When a memory cycle is performed, bits (9:2) of an incoming address index into the Address Memory and select an address tag in Tag 0 Address Memory and Tag 1 Address Memory. The two address fields are read from the Address Memory and compared with the address field (bits 21:10) of the incoming address. If either comparison results in a match, if the corresponding Valid bit is set, and if no address tag parity error is detected, HIT 0 or HIT 1 is asserted. These signals perform the final selection of the words output from the FDM, as illustrated in Figure 2-4.

2.2.3 Cache Operation

When a 22-bit address arrives from the processor or Unibus, bits (9:2) (the index field) are immediately used as an index into the 256 by 30 bit Address Memory, which contains the high order bits (address field) of the addresses presently stored in the Cache and their Valid bits. At the end of the Address Memory access time, two tags are available for use. Each tag consists of a 12-bit address field, a Valid bit, and two parity bits. The two address fields go directly to two comparators, where they are compared with the 12 high order bits of the incoming address. (The stored address tags are checked for correct parity while the address fields are compared. The following discussion assumes that no address tag parity errors are detected.)

2.2.3.1 Read Hit - Assume that one of the address comparisons results in a match and that the corresponding Valid bit is set. This condition is called a "hit" and means that the word requested is in the Fast Data Memory. The appropriate select signal is therefore sent to the Fast Data Memory. Ten bits are required to select one of the 1024 words in Fast Data Memory. Eight of these bits are the same index bits used to index into the Address Memory. These bits select a set of two blocks, one block in each group. Also required is bit 1 of the incoming address, which selects either the high or low word of the blocks within the set. The last selection is provided by the comparison signal from the address field comparators, which determine which of the two groups (or equivalently, which of the two blocks within the set) has the desired data. After this last signal arrives, the data is available from the data memory and will be sent to the processor or Unibus as required.

2.2.3.2 Read Miss - If neither address field from the Address Memory matches the address field of the incoming address, then the requested data is not in fast memory. This is called a miss condition. When the Cache controller determines that there is no match, it must start a Main Memory cycle to fetch the required block. The block address will be the 20 high order bits of the incoming address.

During the Main Memory access time, the Cache controller can decide where to put the incoming data when it arrives. The index field determines which block within a group is replaced. The controller determines in which group the new block will be placed by examining an internally generated Random bit. When the data block arrives from Main Memory, it is written into the selected block of fast memory, while the requested word is passed along to the processor or Unibus. At the same time, the address field of the block is loaded into the corresponding location in Address Memory along with a set Valid bit. (A set Valid bit is loaded into Address Memory whenever the Fast Data Memory is loaded as a result of a read miss.)

2.2.3.3 Write Hit - During a write cycle initiated from the processor or Unibus, the initial sequence of events in the Cache is the same as during a read cycle: the address comes in, the Address Memory is accessed, and the address fields are compared. If the address fields match and the corresponding Valid bit is set, a hit is indicated and the new data is written into the appropriate word or byte of fast memory, as selected by the index, word, and byte fields of the address and the comparator outputs. Since the PDP-11/70 Cache is implemented using write-through, the data is also written into Main Memory. This ensures that the data in the Main Memory and in the Cache are never different.

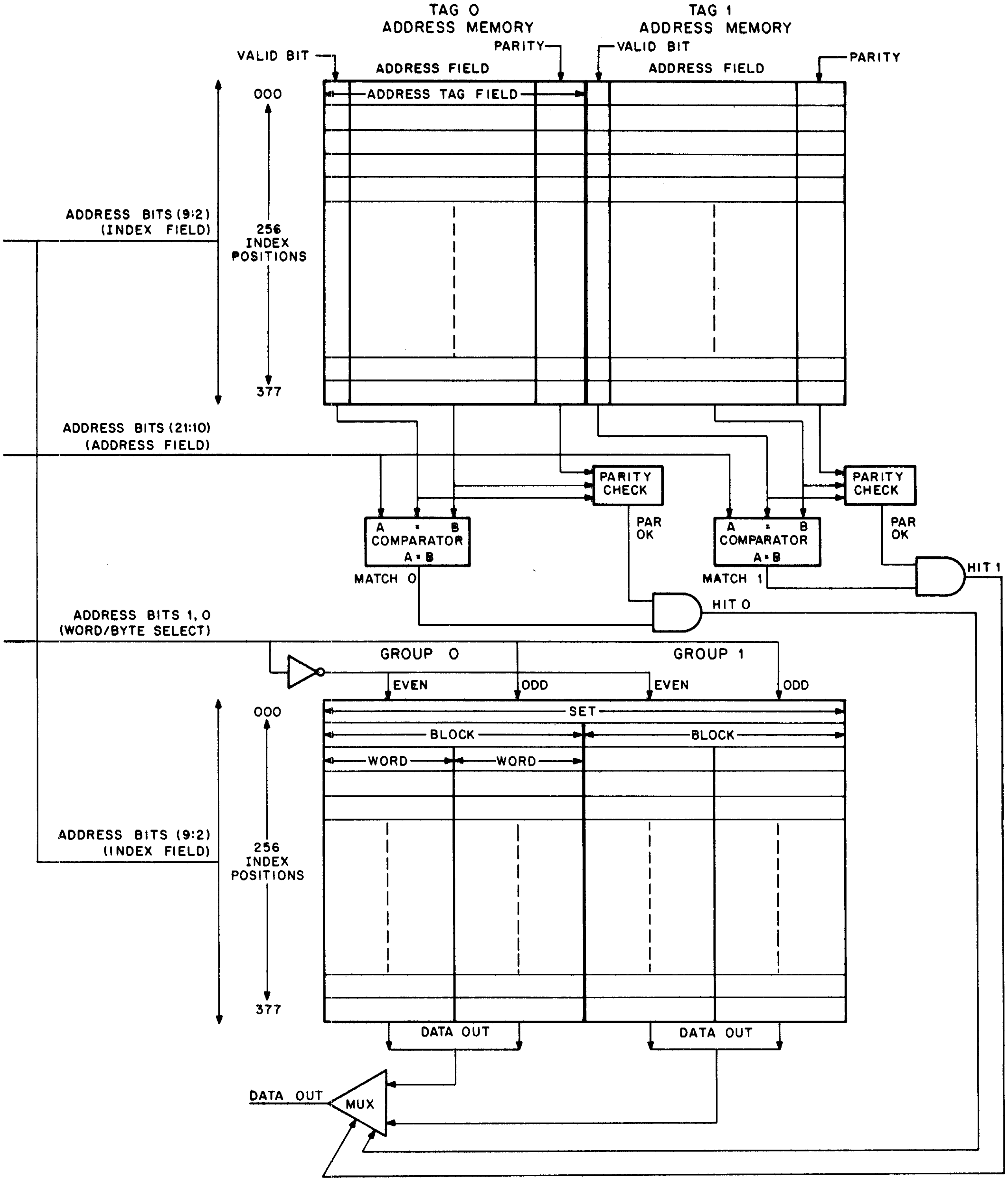


Figure 2-4 PDP-11/70 Cache Simplified Data Path Diagram

2.2.3.4 Write Miss - If, during a write operation from the processor or Unibus, a miss is indicated by the address comparison in the Cache, a write cycle is performed to the specified address in Main Memory. The contents of the Address Memory and the Fast Data Memory are left unaltered.

2.2.3.5 Power-Up Initialization - On power-up, the Cache performs a power-up sequence during which all of the Valid bits in the Address Memory are cleared. This is done because anything stored in the Cache immediately after a power-up must not be construed as valid data. As program execution begins, the density of read misses is high (because of all the negated Valid bits), and data must be fetched from Main Memory. As a result, the FDM gets filled and Valid bits get asserted. This in turn results in fewer misses, i.e., a higher hit rate and greater speed as program execution continues. The time interval required for the memory system to achieve nominal speed is only on the order of 1 ms.

At the same time that the Address Memory Valid bits are negated, all the remaining bits in the Address Memory and FDM are loaded with bit patterns having correct parity. This is to ensure that the bit patterns resident in the Address Memory and FDM upon power-up will not generate parity errors when program execution begins.

2.2.3.6 Overview - It should be apparent to the reader that the mechanisms described in the preceding paragraphs ensure that:

1. The recently used data tends to be in fast memory.
2. The Main Memory always has a correct copy of all data.
3. The same Main Memory location never ends up in two different Fast Data Memory locations at the same time.*

2.3 EXAMPLE OF PDP-11/70 CACHE OPERATION

The following is an example illustrating Cache operations. Assume that the following sequence of events has occurred. The system was powered up, and the machine code listed in Table 2-1 was manually loaded into core memory at the locations specified. Assume that the code was properly loaded and that no verification (read operation) was performed. At this point, none of the Valid bits in the Address Memory of the Cache are asserted. Address 1000 is now loaded into the processor PC, the ENABLE/HALT switch is set to ENABLE, and the START switch is depressed. Program execution, summarized in Table 2-2, begins as follows:

NOTE

The program used in this example is designed to illustrate Cache operations, not a typical use.

1. The processor initiates a fetch of the contents of address 1000.

   a. This results in a miss, because the corresponding Valid bits are unasserted.

   b. The contents of addresses 1000 and 1002 are fetched from Main Memory and loaded into block (index position) 200 of Group 0 of the FDM, as determined by the Random bit.

   c. The address field (0000) is loaded at index position 200 of the corresponding Address Memory along with an asserted Valid bit. At the same time, the contents of address 1000 are sent to the processor.

2. The processor initiates a fetch of the contents of address 1002.

   a. The address field (0000) is compared with the contents of the Address Memory at index position 200. This results in a hit on Group 0 of the FDM. (The match occurs with the tag field loaded at step 1c.)

   b. The requested word is sent to the processor.

3. The processor initiates a fetch of the contents of address location 5000.

   a. The address field (0002) is compared with the contents of Address Memory at index position 200. This results in a miss.

*This condition can occur, however, if the Control Register Force Replacement bits are manipulated.

b. The contents of addresses 5000 and 5002 are fetched from Main Memory and loaded into block (index position) 200 of Group 0 of the FDM as determined by the Random bit. The previous contents of block 200 of Group 0, loaded at step 1c, are overwritten. This is because of the random nature of group selection; the Random bit, as assumed in Table 2-2, happens to be in the same state as it was when the FDM was previously loaded. The corresponding position in Address Memory is also overwritten with the new address field (0002). At the same time, the contents of location 5000 are sent to the processor.

4. The processor initiates a fetch of the contents of location 1004.

   a. The address field (0000) is compared with the contents of the Address Memory at index position 201. Since the Valid bits at this index position are unasserted, this results in a miss.

   b. The contents of address 1004 and 1006 are fetched from Main Memory and loaded into block (index position) 201 of Group 0 of the FDM, as determined by the Random bit.

   c. The address field (0000) is loaded at index position 201 of the corresponding Address Memory along with an asserted Valid bit. At the same time, the contents of address 1004 are sent to the processor.

5. The processor now initiates a write to location 3000. (The data being written was previously fetched from location 5000.)

   a. The address field (0001) is compared with the contents of Address Memory at index position 200. This results in a miss.

   b. The Cache therefore writes the data from the processor into the specified location (3000) in Main Memory. (This illustrates write-through.) The contents of the FDM and Address Memory are left unaltered.

6. The processor initiates a fetch of the contents of location 1006.

   a. The address field (0000) is compared with the contents of the Address Memory at index position 201.

   b. This results in a hit on Group 0 of the FDM. (The match occurs with the address field loaded at step 4c.)

   c. The requested word is sent to the processor.

7. The processor initiates a fetch of the contents of location 1010.

   a. The address field (0000) is compared with the contents of the Address Memory at index position 202. This results in a miss.

   b. The contents of addresses 1010 and 1012 are fetched from Main Memory and loaded into block (index position) 202 of Group 0 of the FDM, as determined by the Random bit.

   c. The address field (0000) is loaded at index position 202 of the corresponding Address Memory along with an asserted Valid bit. At the same time, the contents of address 1010 are sent to the processor.

8. The processor now initiates a fetch of the contents of address location 3000.

   a. The address field (0001) is compared with the contents of Address Memory at index position 200. This results in a miss.

b. The contents of addresses 3000 and 3002 are fetched from Main Memory and loaded into block (index position) 200 of Group 1 of the FDM, as determined by the Random bit.

c. The address field (0001) is loaded at index position 200 of the corresponding Address Memory, along with an asserted Valid bit. At the same time, the contents of address 3000 are sent to the processor.

9. The processor now initiates a fetch of the contents of address location 3002.

   a. The address field (0001) is compared with the contents of the Address Memory at index position 200.

   b. This results in a hit on Group 1 of the FDM. (The match occurs with the tag field loaded at step 8c.)

   c. The requested word is sent to the processor.

10. The processor now initiates a DATIP type fetch of the contents of address location 1012.

    a. The address field (0000) is compared with the contents of Address Memory at index position 202.

    b. This results in a hit on Group 0 of the FDM. (The match occurs with the tag field loaded at step 7c.)

    c. The requested word is sent to the processor.

11. The processor increments the received word and then initiates a DATO to write it back into address location 1012.

    a. The address field (0000) is compared with the contents of Address Memory at index position 202.

    b. This results in a write hit on Group 0 of the FDM. (The match occurs with the tag field loaded at step 7c.)

    c. The Cache performs a write cycle to Main Memory. (This is an illustration of write-through.) It also updates the high word at index position (block) 202 of Group 0 of the FDM. If the write operation had been a DATOB, only the specified byte in the FDM (and Main Memory) would be altered.

12. The processor would now fetch the HALT instruction at address 3004 (read miss), execute it, and halt. It should be clear that if the Random bit is asserted, the contents of locations 3004 and 3006 will be loaded into block 201 of Group 1 of the FDM.

Table 2-1
Example Program
(all numbers in octal notation)

Address Loaded	Machine Code	Index Field	Address Field	Mnemonics	Remarks
001000	013737	200	0000	MOV @#5000@#3000	This program moves the INC instruction at address 5000 to address 3000, then jumps to address 3000, performs the INC instruction, and HALTS
001002	005000			5000	
001004	003000	201		3000	
001006	000137			JMP @#3000	
001010	003000	202		3000	
001012	177776				
.					
.					
.					
003000	DONT CARE	200	0001		
003002	001012				
003004	000000	201		HALT	
.					
.					
.					
005000	005237	200	0002	INC @ #	

Table 2-2
Summary of Cache Operations Example

Processor			Random Bit	Hit /	Fast Data Memory						Address Memory		
					GROUP 0			GROUP 1			TAG 0 Address Memory	TAG 1 Address Memory	
						Contents			Contents				
REF	PC	Operation	(Assumed)	Miss	Block	Low Word	High Word	Block	Low Word	High Word	Address Field Loaded	Address Field Loaded	Remarks
1	1000	Fetch (1000) = 013737	0	Read Miss	200	(1000) = 013737	(1002) = 005000				0000		Fetch MOV instruction
2	1002	Fetch (1002) = 005000	1	Hit on	200		↑				↑ Hit		Fetch source address
3	1004	Fetch (5000) = 005237	0	Read Miss	200	(5000) = 005237	(5002) = xxxxxx				0002		Fetch contents at source address
4	1004	Fetch (1004) = 003000	1	Read Miss				201	(1004) = 003000	(1006) = 000137	0000		Fetch destination address
5	1006	Write 005237 into 3000	0	Write Miss									MOV contents of source to destination address
6	1006	Fetch (1006) = 000137	1	Hit on	201		↑				↑ Hit		Fetch JMP instruction
7	1010	Fetch (1010) = 003000	0	Read Miss	202	(1010) = 003000	(1012) = 177776				0000		Fetch destination address
8	3000	Fetch (3000) = 005237	1	Read Miss				200	(3000) = 005237	(3002) = 001012		0001	JUMP; Fetch INC instruction
9	3002	Fetch (3002) = 001012	0	Hit on				200		↑		↑ Hit	Fetch destination address
10	3004	Fetch (001012) = 177776	1	Hit on	202		↑				↑ Hit		Fetch contents of destination address (DATIP)
11	3004	Write 177777 into 001012	0	Hit on	202	(1010) = unaltered	(1012) = 177777				↑ Hit		INC and restore (DATO)
12	3004	Fetch (3004) = 000000	1	Read Miss				201	(3004) = 000000	(3006) = xxxxxx			Fetch and execute HALT instruction

CHAPTER 3
THEORY OF OPERATION

3.1 SCOPE

This chapter provides a detailed explanation of Cache operation within the PDP-11/70 system. Areas covered include PDP-11/70 data paths, Cache data paths, Cache interfaces, and operational flows for the various operations that the Cache can perform. The latter is the key to a full understanding of the PDP-11/70 Cache.

3.2 PDP-11/70 SYSTEM

Figure 3-1 is a block diagram of the PDP-11/70 System showing the address and data lines which interconnect the functional components of the system. The data lines connecting the Cache to the Main Memory and to the Massbus Controllers are 36 bits wide, and comprise 32 bits of data plus 4 parity bits. The remaining data lines are 16 bits wide.

The Cache, because of its function and position relative to the other functional components of the system, acts as a clearing house for all accesses to Main Memory. Requests for Main Memory access come from three sources: processor, Unibus Map, and Massbus Controllers. When more than one of the above require memory access concurrently, priority is given according to the following structure:

1st Priority: Unibus Map
2nd Priority: Massbus Controllers
3rd Priority: Processor

In addition, concurrent requests for memory access by Massbus Controllers are arbitrated in the Cache.

The address inputs to the Cache are 22 bits wide. The 22-bit address from the processor is derived by mapping the processor's 16-bit virtual address. The 22-bit address from the Unibus Map is derived by mapping the 18-bit Unibus address. The 22-bit address from an MBC is the contents of a Memory Address Register (MAR). (The MAR is an extended register, and requires two Unibus DATO operations by the processor to specify a complete 22-bit address.)

Data can be read from the Cache Fast Data Memory (FDM) only during processor and Unibus Map memory accesses. During MBC memory accesses, the Cache merely performs the required data transfers from the Main Memory Bus to the MBCs and vice versa. It can therefore be said that MBC cycles are not "cached." The reason for this is expained in Paragraph 3.6. Note that because the data lines between the MBCs and the Cache are 36 bits wide, two 16-bit words (plus their associated byte parity bits) can be transferred simultaneously.

A 22-bit address input to the Cache is converted into a Main Memory Bus address by stripping off the two least significant bits of the address. This is done because Main Memory is organized into two word (i.e., double word) blocks. Each double word consists of two 16-bit data words plus their associated parity bits. When data is read from Main Memory, a 36-bit double word is transferred via the Main Memory Bus. However, when data is written into Main Memory, it is written on a byte-by-byte basis. There are lines on the Main Memory Bus which determine which bytes will be operated on. During a read operation, these lines are ignored.

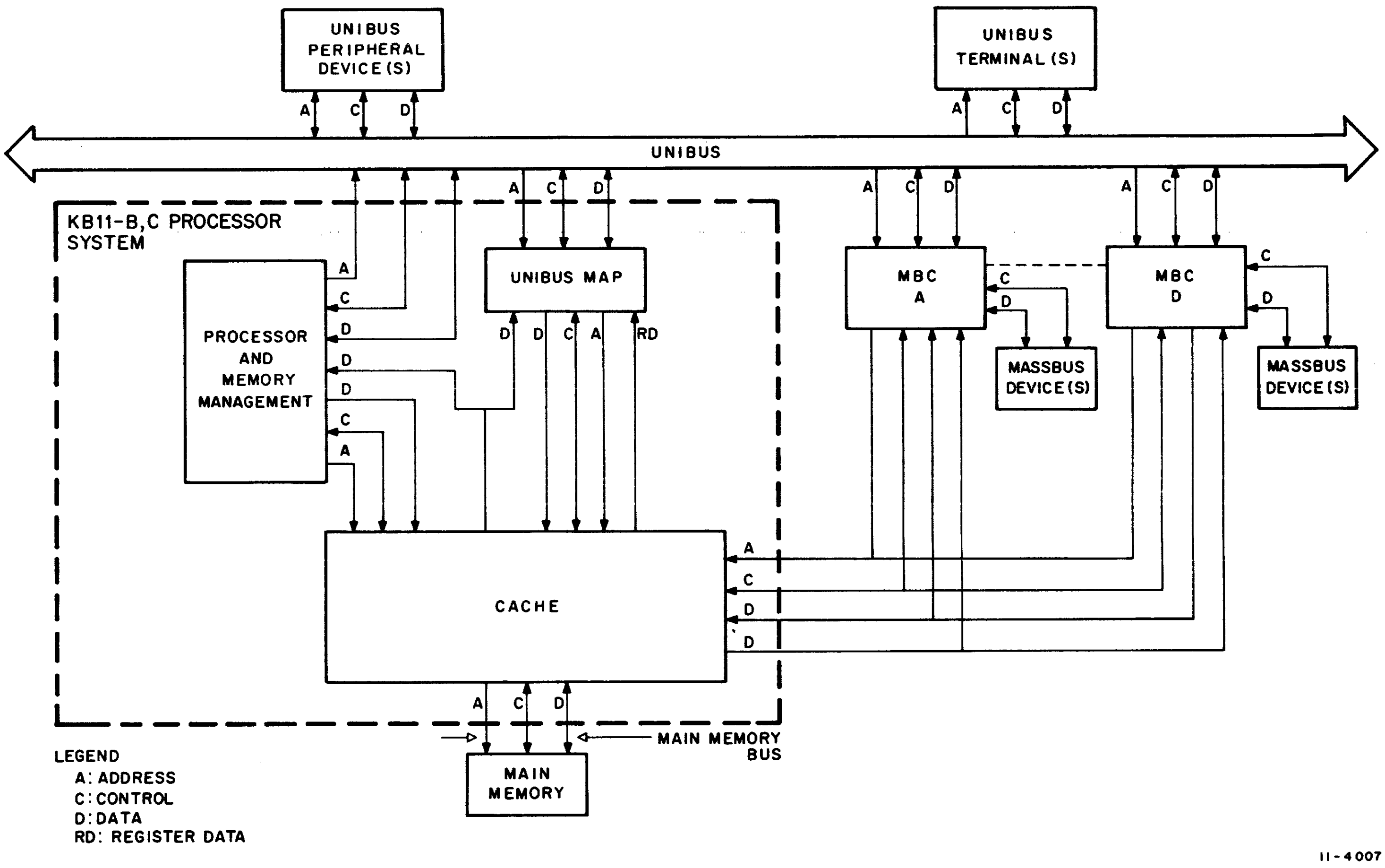


Figure 3-1 PDP-11/70 System

3.2.1 Data Parity

During processor and Unibus Map write operations, data parity bits are generated in the Cache. The parity bits are written into Main Memory along with the data. During a write hit, the data and parity bits are also written into the Fast Data Memory. Parity bits stored in memory (FDM or Main Memory) are treated as data within the memory system. The Cache checks for correct parity when the data is read from memory by the processor or Unibus Map. If a parity error is detected by the Cache, a corresponding bit in the Memory System Error Register is set. If, during a processor read, a parity error is detected on the word requested by the processor, an abort results. However, a parity error on the non-requested word results in a trap (unless traps are disabled). Parity errors during Unibus Map read operations result in a trap. If the parity error is on the requested word, the Unibus parity error line (PB) is asserted.

NOTE

An abort occurs if the processor cannot be supplied with valid data. If a requested word stored in the FDM is found to have bad parity, the Cache fetches the backup copy of the word from Main Memory. If the requested word fetched from Main Memory has incorrect parity, an abort results.

During an MBC cycle, data parity generation and checking is performed in the MBC. MBC parity error handling is thus performed by the MBC's service routine.

3.2.2 Address Parity

The Cache generates parity bits for the address fields which are stored in the Address Memory as a result of a read miss. When the Address Memory is accessed to determine whether a memory cycle is a hit or a miss, the contents of the Address Memory are checked. Detection of a parity error in the Address Memory results in a trap.

The Cache also generates a parity bit for the address and control lines of the Main Memory Bus. The Main Memory checks for correct parity on these lines; if incorrect parity is detected, a parity error line on the Main Memory Bus is asserted. Furthermore, the addressed memory controller will not respond, and a time-out will occur.

3.3 CACHE DATA PATHS

Figure 3-2 is a detailed block diagram of the data paths in the Cache. Each block in the diagram references the location in the engineering drawings where the logic schematics can be found. A detailed description of the block diagram is given in Paragraph 4.2.

3.3.1 Address Paths

Based on arbitration among its three ports, the Cache gates in address and control bits from the selected source. This function is performed by the Address Multiplexer. The incoming address is processed as described below.

Address bits (9:2) index into the Address Memory to select two address tags (one from Tag 0 Address Memory and one from Tag 1 Address Memory). The two tags are checked for correct parity and, at the same time, compared against bits (21:10) of the incoming address. If either address field comparison results in a match, if the corresponding Valid bit is set, and if correct parity is determined, a hit has been detected. This means that the data referenced by the incoming address is currently stored in the FDM.

The address and operation control bits selected by the Address Multiplexer are also used to generate address and control for the Main Memory Bus. Bits (21:02) of the incoming address are driven onto the Main Memory Bus directly. Incoming address bits A01 and A00 are used along with operation control bits C1 and C0 to generate Main Memory Bus control lines MAIN BYTE MASK 3:0 and MAIN C1:0. (Note that the MAIN C1:0 lines are coded differently from the Unibus C1:C0 lines.) In addition, a parity bit is generated for the address and control lines of the Main Memory Bus.

Bits (21:10) of the incoming address are applied to a parity generator along with the internally generated Valid bit. The parity bits generated are applied to the inputs of the Address Memory along with bits (21:10) of the incoming address for possible loading. The Address Memory will be written if either a read miss or a write hit occurs.

3.3.2 Read Data Path

Data is read from Main Memory as 36-bit double words when a read miss is detected. The 36-bit double word is received by Main Memory Bus data receivers and loaded into the Bus Data Register. The even addressed word within the 36-bit double word is gated by the Even Multiplexer to the Cache Data Multiplexer and to the FDM. The Odd Multiplexer performs the same function for the odd addressed word within the 36-bit double word. The data fetched from Main Memory is checked for correct parity at the outputs of the Odd and Even Multiplexers. The Cache Data Multiplexer gates out the word requested to the device that initiated the read. At the same time, the double word is written into FDM Group 0 or Group 1 as selected by the Cache control logic. The index position which is written is determined by bits A09:02 (the index field) of the incoming address.

During a read hit, data is read directly from the FDM. Bits A09:02 of the incoming address index into the FDM. Bit A01 of the incoming address enables either the odd or even word at the indexed location to be output from Group 0 and 1 of the FDM. (Note that the odd and even word outputs of each FDM group are common collectored.) The two words (both odd or even addressed) are checked for correct parity and applied to the Cache Data Multiplexer. The Cache Data Multiplexer selects the word from the group on which the hit occurred and routes it to the device which initiated the read operation.

During an MBC read operation, a 36-bit double word is received in the Cache by Main Memory Bus data receivers and routed to the Massbus Controllers.

3.3.3 Write Data Paths

The data to be written into memory is selected by the write multiplexer, based on whether a processor or Unibus Map cycle is being performed.

NOTE

Processor data is selected by default if a Unibus Map cycle, MBC cycle, or power-up sequence is not being executed. During an MBC cycle or power-up sequence, the write multiplexer outputs are forced to all high. This keeps the data lines stable while data parity is generated.

The outputs of the write multiplexer are applied to the data parity generator, which generates byte parity bits for the 16 bits of data. The data and parity bits are then driven onto the Main Memory Bus data lines. The data is driven on both the low word (MAIN DATA BYTE 1:0) and high word (MAIN DATA BYTE 3:2) lines of the Main Memory Bus. The data can thus be written into either the low word or high word locations in Main Memory.

The output of the write multiplexer and the generated data parity bits are also applied to both the Odd and Even Multiplexers. During a write operation, the Odd and Even Multiplexers select write data as input to the FDM. When a write hit is detected, the write data is therefore available to update the FDM group on which the hit occurred. The FDM is written on a byte-by-byte basis; only the byte(s) referenced by the write operation are altered.

During an MBC write to memory the Cache drives the data from the Massbus Controller onto the Main Memory Bus data lines. The MBCs can transfer single bytes, single words, or double words. If the Cache detects a hit during an MBC write operation, the FDM data on which the hit occurred is invalidated. This is accomplished by negating the Address Memory Valid bit which corresponds to the FDM block on which the hit occurred. The entire FDM block (i.e., four bytes) is thereby invalidated.

3.4 PROCESSOR-CACHE INTERFACE

The signal lines routed between the processor and the Cache may be categorized into two types:

1. Master Timing and Initialization Control

2. Data Transfer Control

The master timing and initialization control lines, originating in the processor, are required by the Cache for its overall operation. These lines route system failure signals, initialization signals, and processor clock signals to which Cache operation is synchronized. For reference, the signals which make up the master timing and initialization control lines are listed in Table 3-1, along with the functions they perform.

The data transfer control lines are active in the transfer of data between the processor and Cache memory. For reference, the signals are listed in Table 3-2, along with their functions.

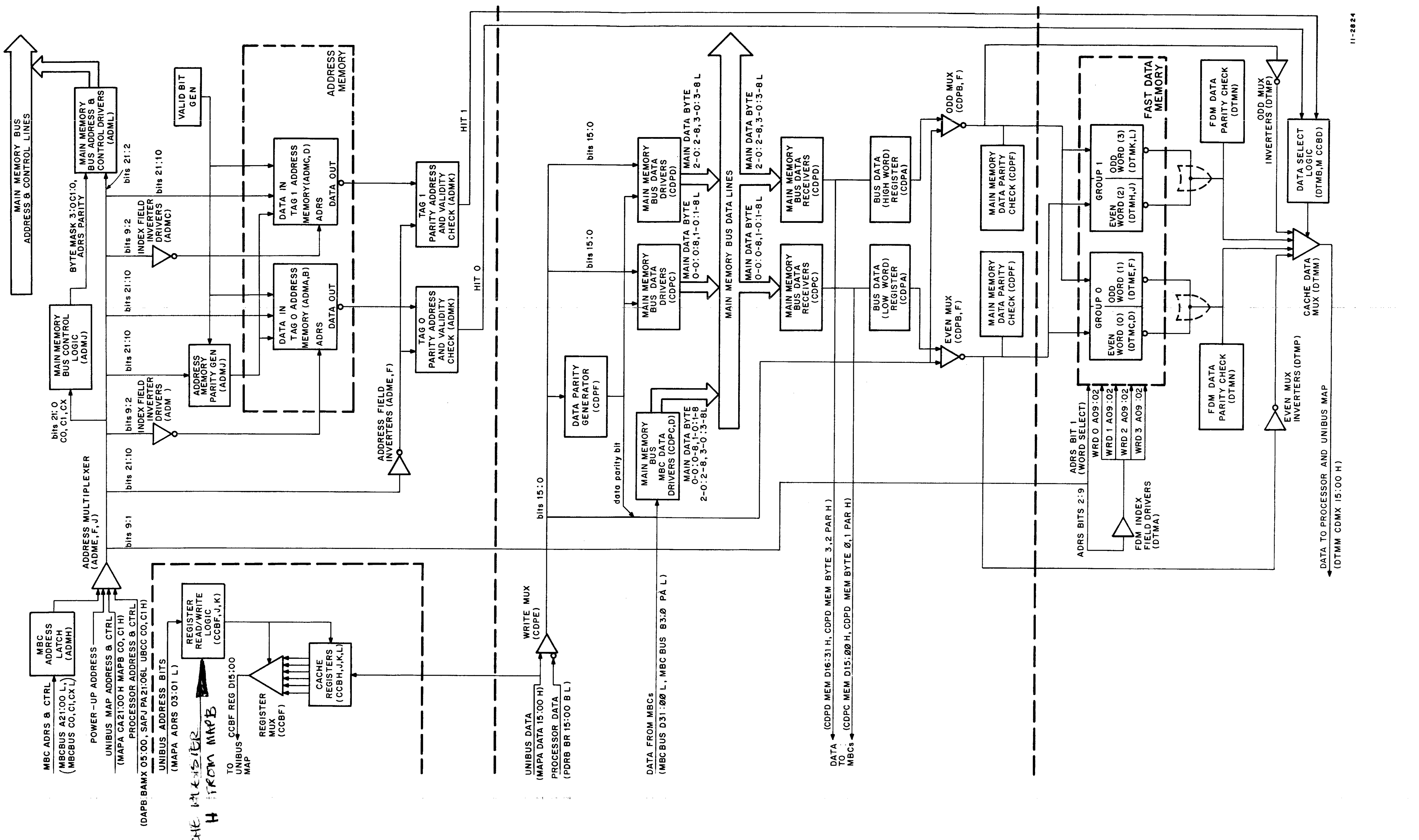

MAIN MEMORY BUS ADDRESS & CONTROL LINES
MAIN MEMORY BUS ADDRESS & CONTROL DRIVERS (ADML)
BYTE MASK 3:0C1:0, ADRS PARITY
bits 21:2
MAIN MEMORY BUS CONTROL LOGIC (ADMJ)
bits 21:0 C0, C1, CX
bits 21:10
bits 9:2
INDEX FIELD INVERTER DRIVERS (ADMC)
INDEX FIELD INVERTER DRIVERS (ADM)
bits 21:10
bits 9:1
ADDRESS MULTIPLEXER (ADME,F,J)
VALID BIT GEN
ADDRESS MEMORY
DATA IN TAG 1 ADDRESS MEMORY(ADMC,D) ADRS DATA OUT
DATA IN TAG 0 ADDRESS MEMORY (ADMA,B) ADRS DATA OUT
ADDRESS MEMORY PARITY GEN (ADMJ)
ADDRESS FIELD INVERTERS (ADME,F)
TAG 1 PARITY ADDRESS AND VALIDITY CHECK (ADMK)
TAG 0 PARITY ADDRESS AND VALIDITY CHECK (ADMK)
HIT 1
HIT 0
MBC ADDRESS LATCH (ADMH)
MBC ADRS & CTRL (MBCBUS A21:00 L, MBCBUS C0,C1,CX L)
POWER-UP ADDRESS
UNIBUS MAP ADDRESS & CTRL (MAPA CA21:00 H MAPB C0, C1 H)
PROCESSOR ADDRESS & CTRL (DAPB BAMX 05:00, SAPJ PA21:06L UBCC C0, C1 H)
UNIBUS ADDRESS BITS (MAPA ADRS 03:01 L)
REGISTER READ/WRITE LOGIC (CCBF,J,K)
REGISTER MUX (CCBF)
CACHE REGISTERS (CCBH,J,K,L)
CCBF REG D15:00
TO UNIBUS MAP
CACHE REGISTER H FROM MAPB
WRITE MUX (CDPE)
UNIBUS DATA (MAPA DATA 15:00 H)
PROCESSOR DATA (PDRB BR 15:00 B L)
bits 15:0
bits 15:0
bits 15:0
DATA PARITY GENERATOR (CDPF)
data parity bit
MAIN MEMORY BUS DATA DRIVERS (CDPD)
MAIN MEMORY BUS DATA DRIVERS (CDPC)
MAIN MEMORY BUS MBC DATA DRIVERS (CDPC,D)
MAIN DATA BYTE 2-0:2-8,3-0:3-8 L
MAIN DATA BYTE 0-0:0-8,1-0:1-8 L
MAIN DATA BYTE 0-0:0-8,1-0:1-8 2-0:2-8,3-0:3-8L
DATA FROM MBCs (MBC BUS D31:00 L, MBC BUS B3:0 PA L)
MAIN MEMORY BUS DATA LINES
MAIN DATA BYTE 2-0:2-8,3-0:3-8 L
MAIN DATA BYTE 0-0:0-8,1-0:1-8 L
MAIN MEMORY BUS DATA RECEIVERS (CDPD)
MAIN MEMORY BUS DATA RECEIVERS (CDPC)
DATA TO MBCs
(CDPD MEM D16:31 H, CDPD MEM BYTE 3,2 PAR H)
(CDPC MEM D15:00 H, CDPD MEM BYTE 0,1 PAR H)
BUS DATA (HIGH WORD) REGISTER (CDPA)
BUS DATA (LOW WORD) REGISTER (CDPA)
ODD MUX (CDPB,F)
EVEN MUX (CDPB,F)
MAIN MEMORY DATA PARITY CHECK (CDPF)
MAIN MEMORY DATA PARITY CHECK (CDPF)
FAST DATA MEMORY
GROUP 1 EVEN WORD (2) (DTMH,J) ODD WORD (3) (DTMK,L)
GROUP 0 EVEN WORD (0) (DTMC,D) ODD WORD (1) (DTME,F)
ADRS BIT 1 (WORD SELECT)
WRD 0 A09:02
WRD 1 A09:02
WRD 2 A09:02
WRD 3 A09:02
ADRS BITS 2:9
FDM INDEX FIELD DRIVERS (DTMA)
FDM DATA PARITY CHECK (DTMN)
FDM DATA PARITY CHECK (DTMN)
ODD MUX INVERTERS (DTMP)
EVEN MUX INVERTERS (DTMP)
DATA SELECT LOGIC (DTMB,M CCBD)
CACHE DATA MUX (DTMM)
DATA TO PROCESSOR AND UNIBUS MAP (DTMM CDMX 15:00 H)
11-2824

Figure 3-2 Cache Data Paths Block Diagram

Table 3-1
Master Timing and Initialization Control Lines

Signal	Function
UBCE AC LO H	This signal is transmitted from the processor to the Cache to notify it that ac input power to one of the power supplies in the system is failing.
UBCE DC LO H	This signal is transmitted to the Cache to notify it that ac input power to one of the power supplies in the system is below the point that guarantees dc outputs to be in regulation. Upon the negation of both of the above signals, the Cache performs its power-up initialization sequence, clearing all the Valid bits in its Address Memory.
UBCE ROM INIT H	This signal is asserted by the processor when it receives DC LO, AC LO, or when a console reset (START switch depressed while in HALT) is performed; it causes the initialization of all the timing state flip-flops in the Cache.
UBCE INIT H	Asserted by the processor upon receipt of AC LO or DC LO or when the console START switch is depressed while the ENABLE/HALT switch is in the ENABLE position. This signal clears the Cache registers.
TIGC TF H	This is the processor free running clock to which Cache operations are synchronized.
TIGC T2B H TIGC T3B H	These are the processor T2 and T3 ROM time states (buffered) used in the Cache to synchronize several particularly critical (timewise) operations. [Examples are generation of CCBC MEM SYNC H and generation of CCBC T2 DLY (1) H.]

Table 3-2
Processor-Cache Data Transfer Control

Signal	Function
DAPB BAMX 05-00 H	These are the six low order bits of the physical address, gated directly from the Bus Address Multiplexer (BAMX) in the processor. (The six low order bits of the processor-generated virtual address are unaltered in generating the 22-bit physical address.) Bit 00 is used to address the FDM during DATOB operations. Bit 01 addresses the FDM to select the desired word within a two-word block. Bits 05 through 02 are part of the index field, and are used to index into the FDM and Address Memory.
SAPJ PA 21-06 H	These are the 16 high order bits of the physical address, generated from the virtual address by the Memory Management. Bits PA09-PA06 are part of the index field, and are used to index into the FDM and Address Memory.

Table 3-2 (Cont)
Processor-Cache Data Transfer Control

Signal	Function
SAPJ PA 21-06 H (cont)	Bits PA21-PA10 comprise the address field, and are compared with the address fields stored at a selected Address Memory index position. They will be loaded into the Address Memory should a read miss occur.
PDRB BR 15-00 B L	These are the outputs of the processor Bus Register (BR), and comprise a 16-bit data word to be stored in memory.
UBCC C1 B H UBCC C0 B H	These are the operation control lines, and indicate the type of operation to be performed, as follows: C1 / C0 / Operation 0 / 0 / DATI 0 / 1 / DATIP 1 / 0 / DATO 1 / 1 / DATOB
RACH BUST H	Asserted by the processor during the "BUST" ROM state to initiate the operation indicated by the C1, C0 bits.
TMCE CACHE BEND H	Asserted by the processor to abort a memory operation initiated by BUST H.
TMCE CONTROL OK H	Asserted by the processor during the "PAUSE" ROM state if it desires to continue with the memory access operation initiated by BUST H.
CCBC MEM SYNC H	Asserted by the Cache and transmitted to the processor at the conclusion of a memory access operation. This signal allows the processor to proceed past T5 of the PAUSE ROM state. During a DATI/DATIP, it also causes the read data to be loaded into the processor's BR register.
DTMM CDMX D15–D00 H	These 16 lines comprise the data word requested by the processor (or Unibus Map) during a DATI/DATIP operation.
DTMM HI BYTE PAR H DTMM LO BYTE PAR H	These signals are the parity bits for the low and high bytes of a requested data word. They are loaded into the processor along with the data word, and are used for console display purposes only.
DTMM BAD PARITY H	Transmitted by the Cache to the processor when a parity error has been detected on a requested data word which has been fetched from Main Memory.
CCBD CP TIMEOUT L	Transmitted by the Cache to the processor when a Main Memory Bus time-out occurs during a CP cycle.
CCBJ PARITY ABORT H	Aborts the processor cycle when good data cannot be given to the processor.

Table 3-2 (Cont)
Processor-Cache Data Transfer Control

Signal	Function
CCBJ PARITY TRAP H	Transmitted to the processor by the Cache to indicate a soft* parity error during a CP cycle or Unibus Map memory (i.e., nonregister) cycle.
PDRH CACHE PERF L	Parity Error flag; this signal is transmitted by the processor to the Cache upon receipt of CCBJ PARITY ABORT H or CCBD CP TIMEOUT, and sets bit 15 (CPU ABORT) or bit 14 (CPU ABORT AFTER LOCK) of the Cache Error Register.
TMCA PERF ACKN L	Transmitted by the processor to the Cache in response to CCBJ PARITY TRAP H; causes its negation.
UBCB UBUS PAR ERR H	Negates CCBJ PARITY TRAP H when processor performs Unibus parity error trap. Sets Error Register bit 09.

*A soft parity error is one which the Cache can recover from without processor intervention and still provide correct data, e.g., parity error in the nonrequested word; parity error in the Address Memory or FDM (if the copy of the requested word in Main Memory is fetched without error).

Processor Cache Protocol

To initiate a data transfer, the processor asserts BUST, and generates an address, operation control bits C1:0, and (if a write is being performed) data. BUST initiates Cache timing. If the processor is performing a BUST-BEND sequence, or if the address generated by the processor is a Unibus address, CACHE BEND is transmitted to the Cache, and brings it to its quiescent state. If the processor is performing a memory access, the Cache receives CONTROL OK from the processor. CONTROL OK indicates that the processor-generated address, control, and data bits are stable and valid, and is treated as a "go ahead" signal by the Cache.

If the processor is performing a read and the Cache detects a hit, the read data is accessed from the FDM. The data is routed to the processor and the Cache asserts MEM SYNC, which causes the data to be loaded into the processor's BR register.

If a read miss is detected, the Cache fetches the data from Main Memory. The Cache routes the requested word to the processor and then asserts MEM SYNC. (Main Memory Bus protocol is described in Paragraph 3.7)

During a write operation, the Cache writes the data into Main Memory and then notifies the processor by asserting MEM SYNC.

The processor must receive MEM SYNC in order to proceed past T5 of the "PAUSE" ROM state.

Figure 3-3 illustrates the processor-Cache protocol.

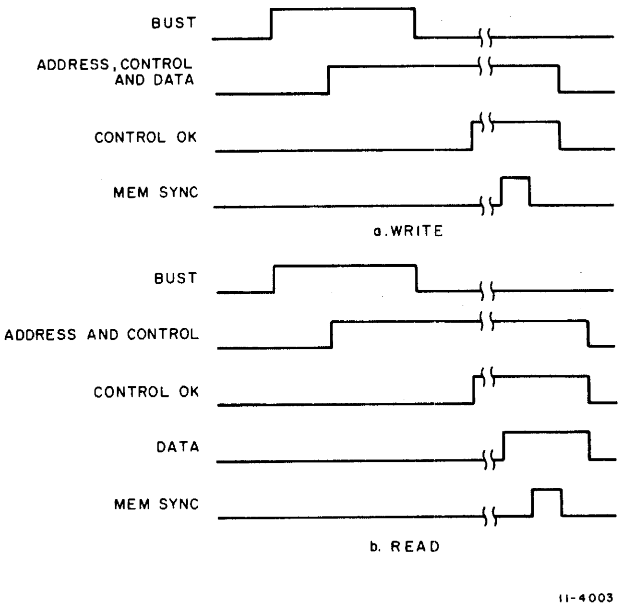


Figure 3-3 Processor – Cache Protocol

3.5 UNIBUS MAP-CACHE INTERFACE

Memory references by Unibus devices are interfaced by the Unibus Map to the Cache. The processor can also access memory via the Unibus and Unibus Map; this is normally done only for maintenance and diagnostic purposes. However, in order to read or write any of the device registers located in the Cache, the processor must do so via the Unibus Map.

For reference, the signals which make up the Unibus Map-Cache interface are listed in Table 3-3 along with their functions.

Unibus Map-Cache Protocol

The Unibus Map interfaces the Unibus data transfer lines to the Cache. When a Unibus device performs a data transfer to or from memory, the device asserts an 18-bit address, operation control bits, and (if performing a write) data on the Unibus. After a deskew delay, the device asserts MSYN.

The Unibus Map generates a 22-bit address from the 18-bit Unibus address and gates the operation control bits and (if performing a write) data to the Cache. When the Unibus Map receives MSYN on the Unibus, it asserts UB REQUEST. UB REQUEST initiates Cache timing. The Cache responds by asserting UB ACKN; this negates UB REQUEST in the Unibus Map. As Cache timing progresses, the address from the Unibus Map is used to access into the FDM and Address Memory.

If the operation is a read and a hit is detected, the requested word is routed from the FDM to the Unibus Map, and the Cache asserts UB DONE. This latches the data in the Unibus Map. If a read miss is detected, the Cache must fetch the data from Main Memory. The Cache routes the requested word to the Unibus Map and then asserts UB DONE. During a write operation, the Cache writes the data into Main Memory and then notifies the Unibus Map by asserting UB DONE. When the Unibus Map receives UB DONE, it terminates its transaction on the Unibus by issuing SSYN. Figure 3-4 illustrates Unibus Map-Cache protocol during read and write operations.

Cache Register Accesses

Unibus Map-Cache protocol during register accesses is similar to normal protocol except for the following points:

1. When the Unibus Map detects that the Unibus address references a Cache device register, it asserts CACHE REG.

2. When CACHE REG is asserted, the Cache uses bits (03:01) of the Unibus address (gated by the Unibus Map) to access the desired register.

3. The data in the accessed register is gated to the Unibus Map, where it is latched by UB DONE if a register read operation is being performed.

4. If a register write operation is being performed, the Unibus data (gated by the Unibus Map) is written into the specified register, and then the Cache asserts UB DONE.

3.6 RH70-CACHE INTERFACE

The Cache handles memory accesses by RH70 Massbus Controllers (MBCs) very differently than processor or Unibus Map memory accesses. MBC

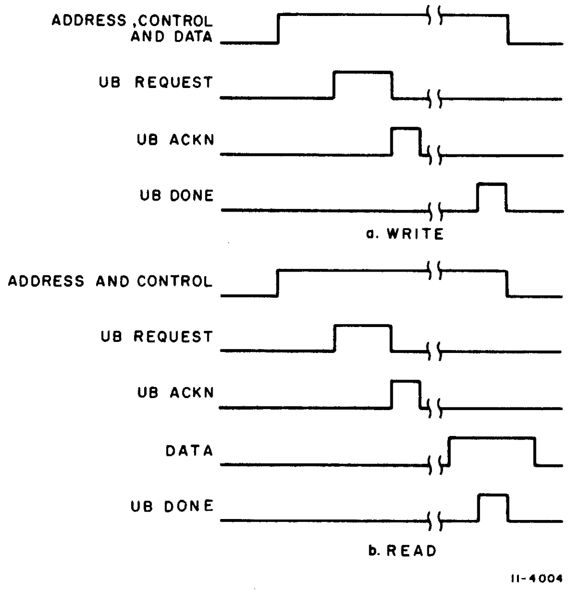


Figure 3-4 Unibus Map - Cache Protocol

Table 3-3
Unibus Map-Cache Interface Signals

Signal	Function
MAPA CA 21–00 H	These 22 lines are the physical address generated by the Unibus Map from the 18-bit Unibus address. Bit 00 is used to address the FDM during DATOB operations. Bit 01 addresses the FDM to select the desired word within a two-word block. Bits 09 through 02 comprise the index field used to index into the FDM and Address Memory. Bits 21 through 10 comprise the address field, and are compared with the address field stored at a selected Address Memory index position. They will be loaded into the Address Memory should a read miss occur. Bits 21 through 02 are also gated onto the Main Memory Bus in case a cycle to Main Memory should be required.
MAPA DATA 15–00 H	These are the Unibus data lines gated by the Unibus Map, and comprise a 16-bit data word to be stored in memory (or written into a Cache device register).
MAPA ADRS 03–01 H	These three address bits are gated from the Unibus by the Unibus Map to access a Cache register.
MAPB C1 H MAPB C0 H	These are the operation control lines gated from the Unibus by the Unibus Map. They indicate the type of operation to be performed, as follows: C1 C0 Operation 0 0 DATI 0 1 DATIP 1 0 DATO 1 1 DATOB
MAPF UB REQUEST (1) L	Asserted by the Unibus Map to initiate the operation indicated by the C1, C0 bits. This occurs after receipt of an address within the Unibus Map response range, and MSYN, on the Unibus.
MAPB CACHE REG H	Asserted by the Unibus Map to indicate that a Cache device register, rather than memory, is being accessed. When a Cache device register is accessed, the Cache utilizes only bits 03 to 01 of the nonrelocated Unibus address gated by the Unibus Map (MAPA ADRS 03–01 H).
CCBC UB ACKN L	Asserted by the Cache when the Unibus cycle has been initiated, to negate UB REQUEST (1) L.

Table 3-3 (Cont)
Unibus Map-Cache Interface Signals

Signal	Function
CCBC UB DONE	Asserted by the Cache to indicate that the Unibus Map memory cycle (or Cache device register access) has been performed. This causes the Unibus Map to accept read data from the Cache and to complete the transaction on the Unibus.
CCBF REG D 15–00	These lines transmit read data from the Cache device registers to the Unibus Map.
DTMM CDMX D15–D00 H	These 16 lines comprise the data word requested from memory by the Unibus Map during a DATI/DATIP operation.
DTMM BAD PARITY H	Asserted by the Cache when a parity error has been detected on a requested data word which has been fetched from Main Memory.
CCBD UB TIMEOUT L	Asserted by the Cache when a time-out has occurred on the Main Memory Bus during a Unibus Map memory access cycle.
MAPB PB DATA H	Transmitted by the Unibus Map to the Cache in response to DTMM BAD PARITY H if the parity error occurred on a valid access, i.e., on an address within the Unibus Map's response range. MAPB PB DATA H inhibits clocking of the Cache Error Address register and sets various bits in the Cache Error Register.

memory accesses always require cycles on the Main Memory Bus, whether or not the operation performed is a read or a write, a hit or a miss. The MBCs never read from or write into the FDM. They only require the Cache to perform the Main Memory Bus protocol needed to access Main Memory. Because the MBCs never read or write into the FDM, it can be said that the MBCs are not cached." The MBCs are handled this way for the following reasons:

1. MBC data transfers differ in their statistical behavior from processor data transfers for which the Cache was designed. If the MBCs "cached," data and codes required by the processor would be swept out of the Cache.

2. Only single words can be output from the FDM, whereas the MBCs are capable of transferring double words.

For reference, Table 3-4 lists all the signals which comprise the RH70-Cache interface, along with their functions.

MBC-Cache Protocol

To initiate a data transfer to or from Main Memory an MBC asserts its request signal (Figure 3-5) CTRLA (or B, or C, or D) REQ.

The Cache arbitrates the simultaneous requests from possibly four MBCs. The protocol proceeds as follows if the request from MBC A is recognized

The Cache transmits SELADRS A to the selected MBC, which responds by gating out address and control lines to the Cache. When the Cache begins executing the MBC cycle, the address and control lines are latched in the Cache. As Cache timing proceeds, the Cache transmits SEL DATA CTRL A to the selected MBC. If the MBC is performing a write operation, this enables it to gate out write data. The Cache then asserts MBC REQ ACKN, which negates the selected MBC's request signal and allows the MBC to alter the address and control bits transmitted to the Cache. The Cache now performs a cycle on the Main Memory Bus. When Main Memory responds with MAIN ACK, the Cache transmits ADRS ACK to the MBC.

If a write to memory is being performed, this terminates the MBC-Cache transaction.

If a read operation is being performed, the Cache routes the double word received from Main Memory to the MBC. When Main Memory asserts MAIN DATA READY, the Cache transmits DATA RDY CNTL "X" to the appropriate MBC. (Although the Cache may be executing some other cycle, it keeps track of which MBC is performing the read operation.)

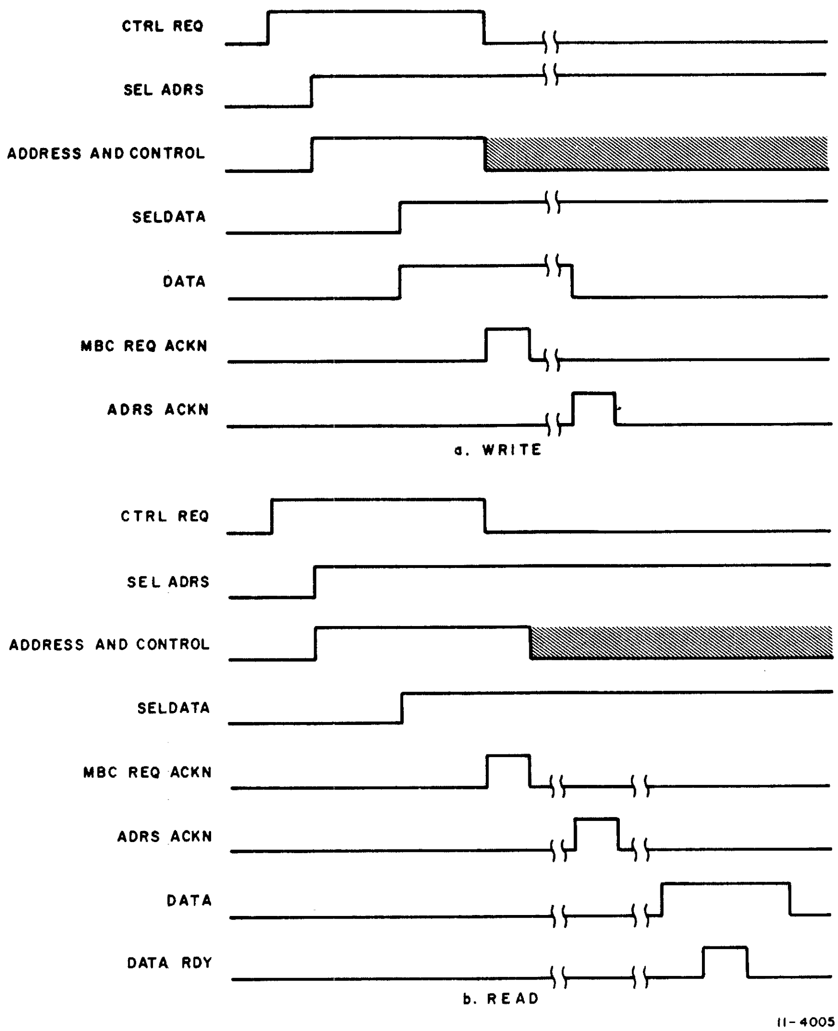


Figure 3-5 RH70 – Cache Protocol

Table 3-4
RH70-Cache Interface Signals

<table>
<tr><th>Signal</th><th>Function</th></tr>
<tr><td>CSTC CTRLA REQ L
CSTC CTRLB REQ L
CSTC CTRLC REQ L
CSTC CTRLD REQ L</td><td>These are the memory access request signals generated by MBC A, B, C, or D, respectively, and transmitted to the Cache Massbus Arbitrator.</td></tr>
<tr><td>CDPJ SELADRS CTRLA H
CDPJ SELADRS CTRLB H
CDPJ SELADRS CTRLC H
CDPJ SELADRS CTRLD H</td><td>One of these signals is asserted by the Cache Massbus Arbitrator to select an MBC requesting memory access. The MBC which receives an asserted SELADRS CTRL X signal is enabled to gate out a memory address and control signals.</td></tr>
<tr><td>MBCBUS A21–A00 L</td><td>These lines transmit a memory address from a selected MBC to the Cache. The address is latched in the Cache MBC Address Register prior to the start of the MBC Main Memory cycle.

Bits 09 through 02 of the address index into the Cache Address Memory (and FDM). Bits 21 through 10 are compared with the address field stored in the Address Memory to determine whether a hit or miss condition exists. (On a write hit, the corresponding data stored in the FDM must be invalidated.)

Bits 21–02 of the address are also gated onto the address lines of the Main Memory Bus in order to perform the required Main Memory Bus operation.</td></tr>
<tr><td>MBC BUS C1, C0, CX L</td><td>These are the operation control lines transmitted from the selected MBC to the Cache. They are latched in the Cache along with the MBC address. C1, C0, and CX determine the type of operation to be performed, as follows:
<table>
<tr><th>C1</th><th>C0</th><th>CX</th><th>Operation</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Read double word</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Read double word</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Write byte</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>Write single word</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>Write double word</td></tr>
</table></td></tr>
<tr><td>CDPJ SEL DATA CTRL A H
CDPJ SEL DATA CTRL B H
CDPJ SEL DATA CTRL C H
CDPJ SEL DATA CTRL D H</td><td>One of these signals is transmitted by the Cache to the corresponding selected MBC. The selected MBC is thereby enabled to gate out data and parity bits to the Cache, if it is performing a write to memory. If it is performing a read, it ignores the SEL DATA signal.</td></tr>
</table>

Table 3-4 (Cont)
RH70-Cache Interface Signals

Signal	Function
MBCBUS D31–D24 L MBCBUS B3 PA L MBCBUS D23–D16 L MBCBUS B2 PA L MBCBUS D15–D08 L MBCBUS B1 PA L MBCBUS D07–D00 L MBCBUS B0 PA L	These are the byte data lines and their corresponding parity bits. An MBC performing a write operation gates out data and parity bits onto these lines when it receives a SEL DATA signal.
CCBE MBC REQ ACKN L	Asserted by the Cache as it begins servicing an MBC request. This signal is transmitted to all MBCs, and notifies the selected MBC to remove its request and enables it to alter the current memory address.
ADML ADRS ACKN L	This signal (received by the Cache from Main Memory as MAIN ACK L) is transmitted to all MBCs. The MBC which last received a SEL DATA signal from the Cache is thereby notified that: 1. Main Memory is responding. 2. If a write to memory is being performed, the current MBC-Cache write data transaction is now terminated.
CDPD MEM D31–D24 H CDPD MEM BYTE 3 PAR H CDPD MEM D23–D16 H CDPD MEM BYTE 2 PAR H CDPC MEM D15–D08 H CDPC MEM BYTE 1 PAR H CDPC MEM D07–D00 H CDPC MEM BYTE 0 PAR H	These are the data lines and their corresponding parity bits received from the Main Memory Bus in the Cache, and then transmitted to the MBCs.
CDPK DATA RDY CNTL A H CDPK DATA RDY CNTL B H CDPK DATA RDY CNTL C H CDPK DATA RDY CNTL D H	When the Cache receives DATA READY from Main Memory during an MBC memory (read) access operation, it transmits DATA RDY CNTL "X" H to the MBC which initiated the read. DATA RDY CNTL X loads the Main Memory data into MBC X and terminates the MBC Cache transaction.
CCBD MBC TIMEOUT L	Asserted by the Cache when an MBC cycle results in a time-out on the Main Memory Bus.

3.7 MAIN MEMORY BUS

The Main Memory Bus interfaces the Cache with the Main Memory. The Main Memory Bus is a defined bus with bidirectional data lines. The address and control lines of the bus are unidirectional and can only be asserted by the Cache. The Cache is thus the sole master of the Main Memory Bus; only the Cache can initiate data transfers on the Main Memory Bus.

The Cache can perform two types of memory operations: a read and a write. When a read operation is performed, the Main Memory transmits a 36-bit double word to the Cache. A write operation, however, can be performed on specified bytes or words within an addressed double word.

The Main Memory Bus is made up of four type BC06R cables. Two cables carry the Main Memory Bus data lines, while the other two carry the Main Memory Bus address and control lines. Each cable contains 40 conductors. Alternating conductors and the cable shield are grounded to reduce crosstalk; this leaves 20 conductors in each cable to carry the Main Memory Bus interface signals. The Main Memory Bus signals are asserted low (~0.4 V), and are high (~3.2 V) when negated.

For reference, Table 3-5 lists all the Main Memory Bus signals and their corresponding cable conductor number and connector pin number, while Table 3-6 describes the functions of these signals.

Main Memory Bus Protocol

To initiate a memory operation, the Cache performs the following steps (refer to Figure 3-6):

1. The Cache places the address of the desired double word on the Main Memory Bus address lines (MAIN A24:02)

2. The Cache places six control bits (MAIN C1:0 and BYTE MASK 3:0) defining the operation to be performed on the Main Memory Bus.

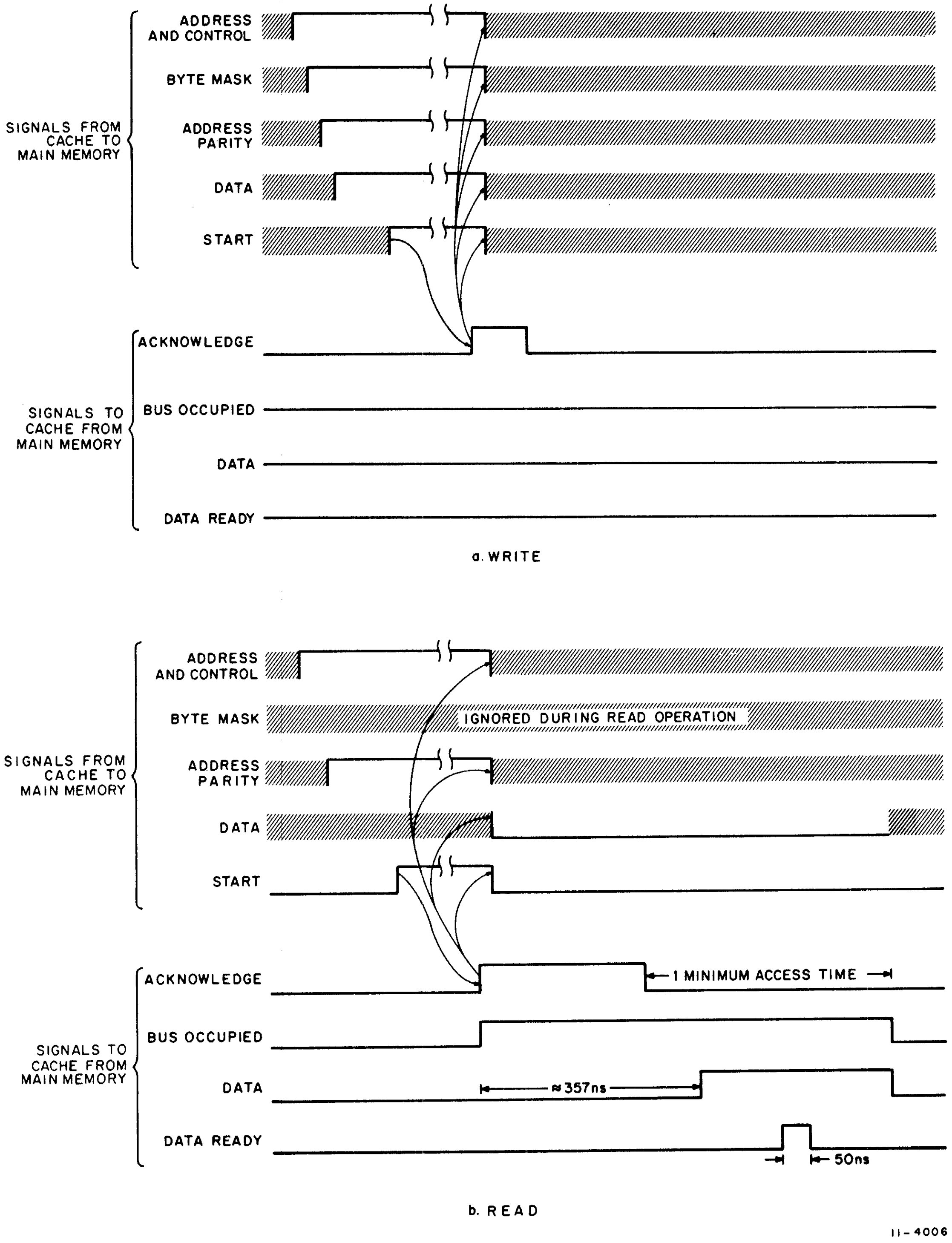
3. The Cache places a parity bit corresponding to the above address and control lines on the Main Memory Bus.

   NOTE

   Address lines MAIN A24:22 and MAIN C0 are always maintained in the negated state in the PDP-11/70. The Cache selects a read operation by negating MAIN C1 and selects a write operation by asserting MAIN C1. During a write operation, the four Byte Mask bits determine which of the four bytes within the addressed double word will be operated on. If BYTE MASK "X" is asserted, byte "X" of the double word will be written. The Byte Mask bits are negated by the Cache during a read operation; this is done only to ensure that the lines remain stable while Main Memory checks for correct parity for the address and control lines. The Byte Mask bits are otherwise ignored by the Main Memory during a read operation.

4. If a write operation is to be performed, the Cache gates out data onto the Main Memory Bus data lines (MAIN DATA BYTE 3-8:0-0). The Cache must wait until the bus data lines become unoccupied (MAIN BOCC negated) before it can gate out the data.

5. After an access and deskew delay for the address, control, parity, and data lines, the Cache issues MAIN START.



a. WRITE

b. READ

Figure 3-6 Cache – Main Memory Protocol

Table 3-5
Main Memory Bus Signals

Signal	Function
MAIN A (21:02) L	This is the 20-bit address of a 2-word block located in Main Memory. They are the high order bits of a 22-bit physical address gated onto the Main Memory Bus by the Cache. (Main Memory Bus address lines MAIN A24:22 are not used by the Cache, and are always maintained in the negated state.)
MAIN BYTE MASK 3 L MAIN BYTE MASK 2 L MAIN BYTE MASK 1 L MAIN BYTE MASK 0 L	These bits define which of the 4 bytes within the block addressed by MAIN A (21:02) will be operated on. There is a one-to-one correspondence between bytes 3 to 0 and byte mask 3 to 0. If MAIN BYTE MASK "X" is asserted, byte "X" will be operated on. The byte mask signals are derived by the Cache from the two low order bits of an incoming address, operation control bits C1 and C0, and the CX bit if an MBC operation is being performed. The Main Memory ignores the mask bits on all DATI/DATIP operations.
MAIN C (1:0) L	These bits, gated from the Cache, determine which operation is to be performed by Main Memory. They are decoded as follows: C1 C0 Operation 0 0 Read 0 1 Not used 1 0 Write 1 1 Exchange (not used by PDP-11/70 Cache)
MAIN A PAR L	This is an odd parity bit generated by the Cache (on drawing ADMJ) for the 26-bit control word consisting of MAIN A (21:02), MAIN C(1:0) and MAIN BYTE MASK (3:0). This bit is received and checked by the Main Memory.
MAIN START L	Asserted by the Cache to initiate the Main Memory cycle designated by the MAIN C (1:0) bits on the Main Memory address locations designated by the address and Byte Mask bits.
MAIN PAR ERR L	Asserted by the Main Memory if it detects a parity error in the 26-bit address and control word described above when START is asserted.
MAIN ACK L	This signal is asserted by the addressed memory controller within Main Memory when it has actually started execution of the commanded memory cycle. Receipt of MAIN ACK in the Cache allows it to alter MAIN A (21:02), MAIN C (1:0), MAIN BYTE MASK (3:0), and MAIN ADDR PAR lines on the Main Memory Bus, and to negate MAIN START. If a write operation was just initiated, MAIN ACK indicates that the Main Memory Bus transaction is now terminated.
MAIN BOCC L	During a read operation, this signal is asserted coincidentally with MAIN ACK by the active memory controller within Main Memory, and is kept asserted until the data is removed from the Main Memory Bus data lines [MAIN DATA BYTE (0-0:0-8; 1-0:1-8; 2-0:2-8; 3-0:3-8)]. The Cache may gate data onto the Main Memory Bus data lines only when MAIN BOCC is not asserted.
MAIN DATA BYTE (0-0:0-8) L MAIN DATA BYTE (1-0:1-9) L MAIN DATA BYTE (2-0:2-8) L MAIN DATA BYTE (3-0:3-9) L	These are the data and data parity lines of the Main Memory Bus. Odd parity is utilized. MAIN DATA BYTE 1-9 and 3-9 L are not used in the PDP-11/70 implementation. The MAIN DATA BYTE lines are organized as follows: MAIN DATA BYTE (0-0:0-7) – Byte 0 data MAIN DATA BYTE 0-8 – Byte 0 parity MAIN DATA BYTE (1-0:1-7) – Byte 1 data MAIN DATA BYTE 1-8 – Byte 1 parity MAIN DATA BYTE (2-0:2-7) – Byte 2 data MAIN DATA BYTE 2-8 – Byte 2 parity MAIN DATA BYTE (3-0:3-7) – Byte 3 data MAIN DATA BYTE 3-8 – Byte 3 parity During a write operation, the Cache gates out data and data parity bits onto these lines while MAIN BOCC is negated, and then asserts MAIN START. Bytes are written into memory along with their parity bits. Which bytes are written is determined by the MAIN BYTE MASK bits. The parity bits are generated in the Cache if a processor or Unibus Map write cycle is performed. The parity bits are received from an MBC if an MBC write to memory operation is being performed. During a read operation, Main Memory brings up all four bytes and their corresponding parity bits and places the information on the MAIN DATA BYTE lines.
MAIN DATA READY L	This signal is asserted by Main Memory during a read operation, after it has placed data on the MAIN DATA BYTE lines, to indicate to the Cache that the requested data is available.
MAIN AC LOW L	Asserted by Main Memory to inform the processor that the ac power input to a Main Memory power supply is failing.
MAIN DC LOW L	Asserted by the Cache or Main Memory to inform the rest of the system that input power to a power supply somewhere in the system is below the point that guarantees dc outputs to be in regulation.

Table 3-6
Memory Bus Signal Pin Connections

Memory Bus Cable Conductor No.	Bus Master or Memory Controller "OUT" Connector Pin	Data Cable A	Data Cable B	Address Cable A	Address Cable B
1	B	MAIN DATA BYTE 0–0 L	MAIN DATA BYTE 2–0 L	MAIN A02 L	MAIN A22 L
2	A	Gnd	Gnd	Gnd	Gnd
3	D	MAIN DATA BYTE 0–1 L	MAIN DATA BYTE 2–1 L	MAIN A03 L	MAIN A23 L
4	C	Gnd	Gnd	Gnd	Gnd
5	F	MAIN DATA BYTE 0–2 L	MAIN DATA BYTE 2–2 L	MAIN A04 L	MAIN A24 L
6	E	Gnd	Gnd	Gnd	Gnd
7	J	MAIN DATA BYTE 0 3 L	MAIN DATA BYTE 2–3 L	MAIN A05 L	MAIN A PAR L
8	H	Gnd	Gnd	Gnd	Gnd
9	L	MAIN DATA BYTE 0–4 L	MAIN DATA BYTE 2–4 L	MAIN A06 L	MAIN C0 L
10	K	Gnd	Gnd	Gnd	Gnd
11	N	MAIN DATA BYTE 0–5 L	MAIN DATA BYTE 2–5 L	MAIN A07 L	MAIN C1 L
12	M	Gnd	Gnd	Gnd	Gnd
13	R	MAIN DATA BYTE 0–6 L	MAIN DATA BYTE 2–6 L	MAIN A08 L	MAIN BYTE MASK 00 L
14	P	Gnd	Gnd	Gnd	Gnd
15	T	MAIN DATA BYTE 0–7 L	MAIN DATA BYTE 2–7 L	MAIN A09 L	MAIN BYTE MASK 01 L
16	S	Gnd	Gnd	Gnd	Gnd
17	V	MAIN DATA BYTE 0–8 L	MAIN DATA BYTE 2–8 L	MAIN A10 L	MAIN BYTE MASK 02 L
18	U	Gnd	Gnd	Gnd	Gnd
19	X	MAIN DATA BYTE 1–0 L	MAIN DATA BYTE 3–0 L	MAIN A11 L	MAIN BYTE MASK 03 L
20	W	Gnd	Gnd	Gnd	Gnd
21	Z	MAIN DATA BYTE 1–1 L	MAIN DATA BYTE 3–1 L	MAIN A12 L	MAIN ACK L
22	Y	Gnd	Gnd	Gnd	Gnd
23	BB	MAIN DATA BYTE 1–2 L	MAIN DATA BYTE 3–2 L	MAIN A13 L	MAIN PAR ERR L
24	AA	Gnd	Gnd	Gnd	Gnd
25	DD	MAIN DATA BYTE 1–3 L	MAIN DATA BYTE 3–3 L	MAIN A14 L	MAIN START L
26	CC	Gnd	Gnd	Gnd	Gnd
27	FF	MAIN DATA BYTE 1–4 L	MAIN DATA BYTE 3–4 L	MAIN A15 L	MAIN BOCC L
28	EE	Gnd	Gnd	Gnd	Gnd
29	JJ	MAIN DATA BYTE 1–5 L	MAIN DATA BYTE 3–5 L	MAIN A16 L	MAIN MARGIN 0 L
30	HH	Gnd	Gnd	Gnd	Gnd
31	LL	MAIN DATA BYTE 1–6 L	MAIN DATA BYTE 3–6 L	MAIN A17 L	MAIN MARGIN 1 L
32	KK	Gnd	Gnd	Gnd	Gnd
33	NN	MAIN DATA BYTE 1–7 L	MAIN DATA BYTE 3–7 L	MAIN A18 L	MAIN MARGIN 2 L
34	MM	Gnd	Gnd	Gnd	Gnd
35	RR	MAIN DATA BYTE 1–8 L	MAIN DATA BYTE 3–8 L	MAIN A19 L	MAIN AC LOW L
36	PP	Gnd	Gnd	Gnd	Gnd
37	TT	MAIN DATA BYTE 1–9 L	MAIN DATA BYTE 3–9 L	MAIN A20 L	MAIN DC LOW L
38	SS	Gnd	Gnd	Gnd	Gnd
39	VV	MAIN DATA READY L	SPARE	MAIN A21 L	SPARE
40*	UU	Gnd	Gnd	Gnd	Gnd

*(Cable Shield)

Response of Main Memory
Each memory controller in Main Memory checks for correct parity in the address and control lines. If a parity error is detected when MAIN START is received, MAIN PAR ERR is asserted on the bus. If the addressed memory controller detects a parity error, execution of the memory cycle is inhibited and a time-out results.

Write Operation - When Main Memory begins executing the requested memory cycle, it latches Main Memory Bus address, control, and data lines, and asserts MAIN ACK on the bus. With the required information latched in Main Memory, active participation by the Cache is no longer necessary. When the Cache receives MAIN ACK, it is notified that the Main Memory transaction is terminated. MAIN ACK negates MAIN START in the Cache. When MAIN ACK becomes negated, the Cache can again assert MAIN START if address, control, and data (if applicable) have been sufficiently deskewed.

Read Operation - When Main Memory begins executing the requested memory cycle, it latches Main Memory Bus address and control lines, and asserts MAIN ACK and MAIN BOCC. After the Main Memory access delay, read data is placed on the bus. After a data deskew delay, the Main Memory asserts MAIN DATA READY for approximately 50 ns. The Main Memory then removes the read data from the bus, and simultaneously negates MAIN BOCC.

If the Cache is performing a processor or Unibus Map cycle, the DATA READY signal latches the Main Memory Bus data in the Bus Data Register of the Cache. If the Cache is performing an MBC cycle, the Main Memory Bus data and a data ready signal are routed by the Cache to the MBC performing the read from memory.

Initiation of Overlapped Cycles
When performing an MBC read operation, the Cache does not need to wait for the assertion of DATA READY before it can initiate the next Main Memory cycle.

If the next cycle is a read operation, the Cache can assert MAIN START as soon as MAIN ACK is negated, providing that the Main Memory Bus address and control lines have been stable for the required time period. Thus, two MBC read operations may be performed back to back; this is termed "stacking MBC reads." When MBC reads are stacked, the Cache routes the DATA READY signals from Main Memory to the appropriate MBC.

If the next cycle is a write operation, the Cache must also wait for MAIN BOCC to become unasserted, indicating that Main Memory is no longer driving the bidirectional data lines of the Main Memory Bus. The Cache may then assert write data on the bus, wait the required data deskew delay, and issue MAIN START.

3.8 OPERATIONAL FLOWS
This paragraph provides a dynamic description of Cache operations. Flowcharts and flowchart descriptions are provided for each type of operation that the PDP-11/70 Cache can perform. The flowcharts illustrate the relationships and interdependence of the various Cache functions. Specific references to the Cache schematic diagrams are made for each discrete function, allowing direct use of the flowcharts, along with the schematics, in troubleshooting the Cache hardware.

Only five different symbols are used in the flowcharts; these are defined in Figure 3-7.

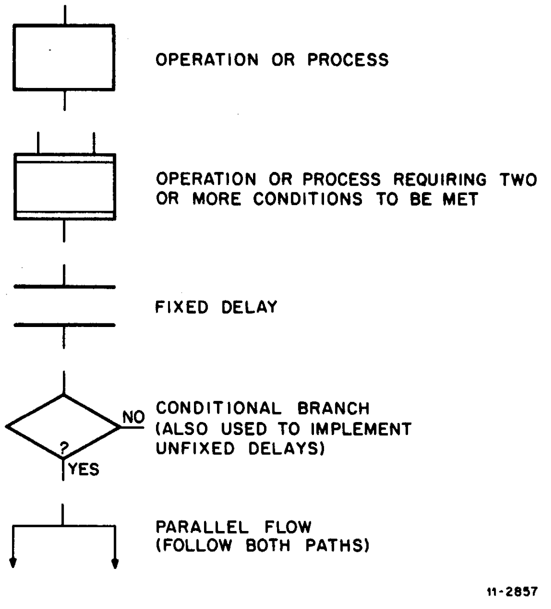

OPERATION OR PROCESS
OPERATION OR PROCESS REQUIRING TWO OR MORE CONDITIONS TO BE MET
FIXED DELAY
NO
?
YES
CONDITIONAL BRANCH (ALSO USED TO IMPLEMENT UNFIXED DELAYS)
PARALLEL FLOW (FOLLOW BOTH PATHS)
11-2857

Figure 3-7 Flowchart Symbol Definitions

3.8.1 Processor Read Hit

Figure 3-8 is a flowchart illustrating Cache operation during a processor read hit. The processor may initiate a data transfer only when it is in a "BUST" ROM state. When the processor performs a read operation, it generates a 16-bit virtual address and control bits C1 and C0. Memory Management converts the virtual address to a 22-bit physical address which is routed to the Cache. During the BUST ROM state, the processor asserts BUST; this causes "BUST" HOLD to be asserted in the Cache. A CP cycle will be initiated by the Cache *if*:

1. The Cache is not presently servicing the request of some other device.
2. The Cache is not presently waiting to execute the write portion of a DATIP initiated by some other device.
3. There are no other requests pending (i.e., PRE UBUS or PRE MBC is not asserted).

If the above conditions are satisfied when (or while) BUST is asserted, or even if the above conditions are satisfied when only BUST HOLD is asserted, the Cache asserts CP CYCLE and LOCK. LOCK indicates that the Cache is presently "locked" into an operating cycle (CP CYCLE in this case) and that no other requests will be serviced until the present cycle is completed. CP CYCLE causes the address generated by the Memory Management to be gated into the Cache by the Address Multiplexer. This address is processed in the Cache and, at the same time, gated to the Main Memory Bus (along with control bits), in case a slow cycle to Main Memory will be required. Incoming address bits (9:2) address the FDM to select data to be read. Incoming address bits (9:2) also address the Address Memory. Incoming address bits (21:10) are checked against the contents of the Address Memory to determine whether the contents of the address referenced are currently stored in the Cache. HIT 0 or HIT 1 will be asserted if the data being requested is in the FDM. Since this paragraph discusses processor read hits, assume HIT 0 is asserted and that an odd word address (XXXXXX2 or XXXXXX6) is being read. Address bit 1 = 1 (odd word address) causes the odd areas of the FDM to be enabled; therefore, an odd addressed word is output from each group of the FDM. HIT 0 asserted causes the Cache Data Multiplexer to gate out only the odd addressed word from Group 0 of the FDM; this word is routed to the processor.

When LOCK is asserted, a timing sequence is initiated in the Cache. If the processor does not wish to abort the read operation, it asserts CONTROL OK, which, when received by the Cache, allows the generation of MEM SYNC and the negation of BUST HOLD at T180 of the Cache timing sequence. BUST HOLD negated prevents the Cache from responding twice to the same processor BUST cycle. MEM SYNC is routed to the processor; receipt of MEM SYNC allows the processor to continue past time state T5 of the "PAUSE" ROM state. MEM SYNC also causes the data from the Cache to be loaded into the processor's BR.

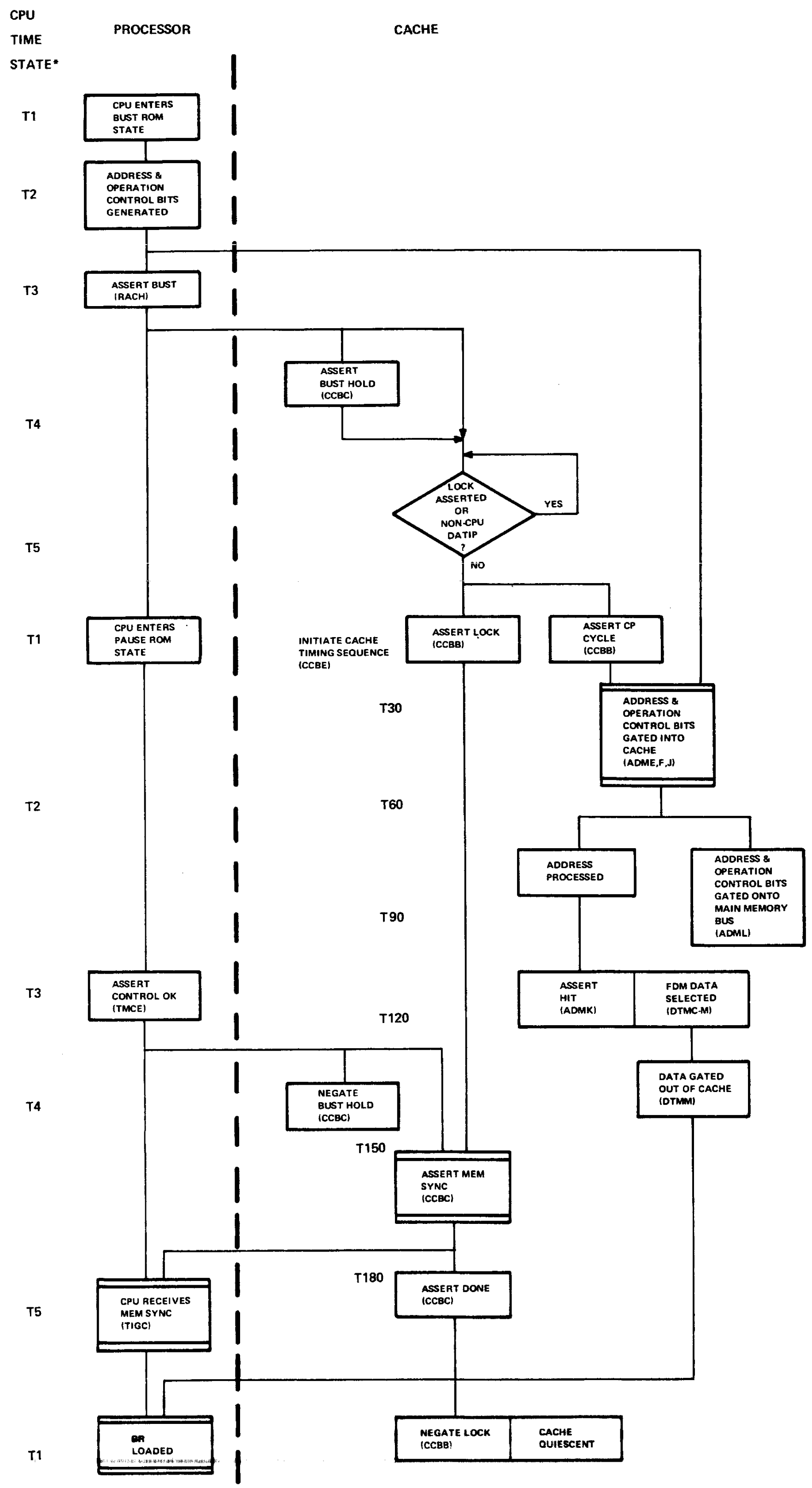
At T180 of the Cache timing sequence, DONE is asserted in the Cache. This negates LOCK and thereby brings the Cache to its quiescent state. With LOCK negated, the Cache can begin servicing other requests for memory access.

3.8.2 Processor Read Miss

Figure 3-9 is a flowchart illustrating Cache operation during a processor read miss cycle. The processor may initiate a data transfer only when it is in a "BUST" ROM state. When the processor performs a read operation, it generates a 16-bit virtual address and control bits C1 and C0. Memory Management converts the virtual address to a 22-bit physical address which is routed to the Cache. During the "BUST" ROM state, the processor asserts BUST; this causes BUST HOLD to be asserted in the Cache. A CP cycle will be initiated by the Cache *if*:

1. The Cache is not presently servicing the request of some other device (i.e., LOCK is not asserted).
2. The Cache is not presently waiting to execute the write portion of a DATIP initiated by some other device.
3. There are no other requests pending (i.e., PRE UBUS or PRE MBC is not asserted).

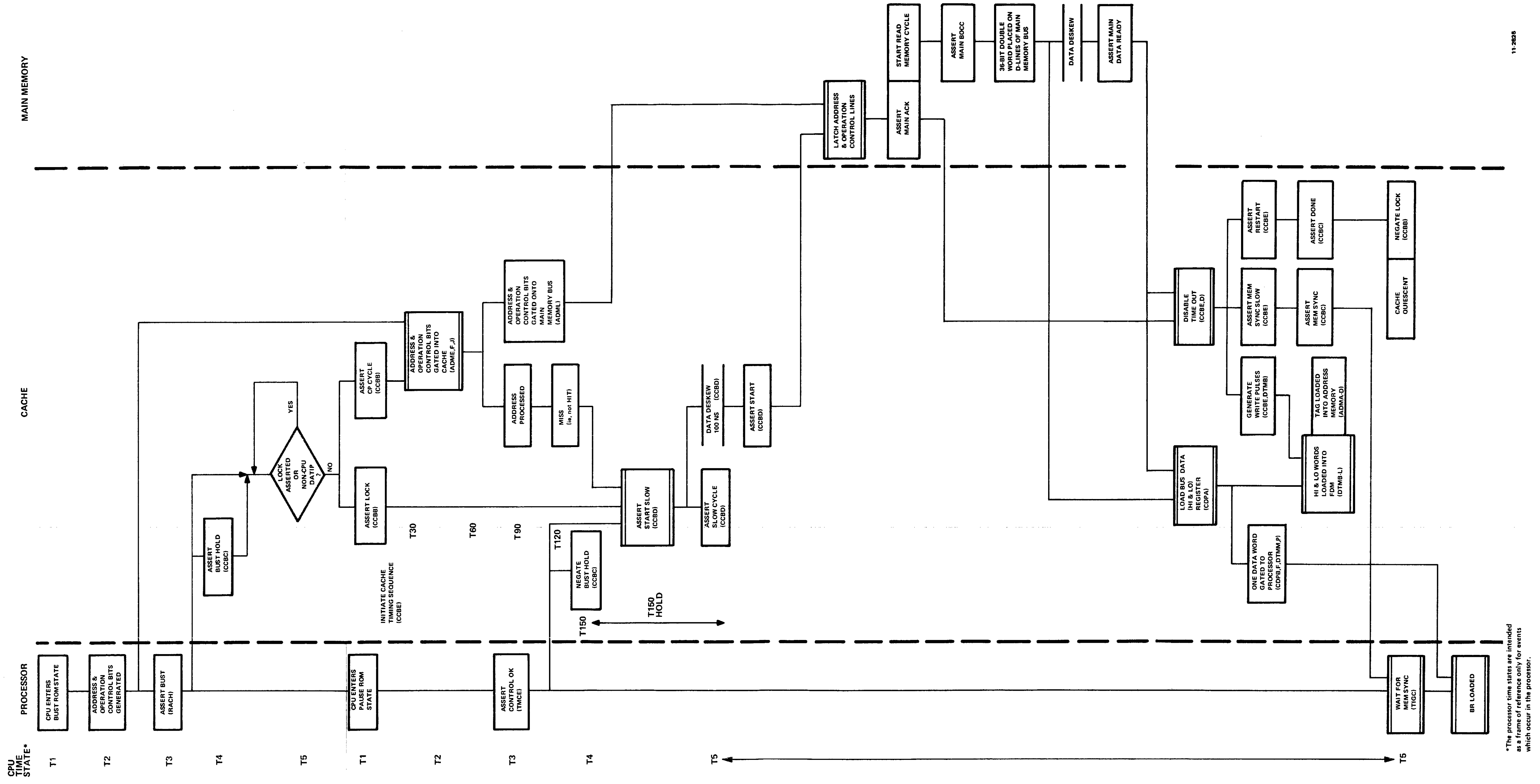
If the above conditions are satisfied when (or while) BUST is asserted, or even if the above conditions are satisfied when only BUST HOLD is asserted,


CPU
TIME
STATE*
PROCESSOR
CACHE
T1
CPU ENTERS BUST ROM STATE
T2
ADDRESS & OPERATION CONTROL BITS GENERATED
T3
ASSERT BUST (RACH)
ASSERT BUST HOLD (CCBC)
T4
LOCK ASSERTED OR NON-CPU DATIP ?
YES
NO
T5
T1
CPU ENTERS PAUSE ROM STATE
INITIATE CACHE TIMING SEQUENCE (CCBE)
ASSERT LOCK (CCBB)
ASSERT CP CYCLE (CCBB)
T30
ADDRESS & OPERATION CONTROL BITS GATED INTO CACHE (ADME,F,J)
T2
T60
ADDRESS PROCESSED
ADDRESS & OPERATION CONTROL BITS GATED ONTO MAIN MEMORY BUS (ADML)
T90
T3
ASSERT CONTROL OK (TMCE)
T120
ASSERT HIT (ADMK)
FDM DATA SELECTED (DTMC-M)
T4
NEGATE BUST HOLD (CCBC)
DATA GATED OUT OF CACHE (DTMM)
T150
ASSERT MEM SYNC (CCBC)
T180
ASSERT DONE (CCBC)
T5
CPU RECEIVES MEM SYNC (TIGC)
T1
BR LOADED
NEGATE LOCK (CCBB)
CACHE QUIESCENT

*The processor time states are intended as a frame of reference only for events which occur in the processor.

11-2830

Figure 3-8 Processor Read Hit


MAIN MEMORY
CACHE
PROCESSOR
CPU TIME STATE*
T1
T2
T3
T4
T5
T1
T2
T3
T4
T5
T5
CPU ENTERS BUST ROM STATE
ADDRESS & OPERATION CONTROL BITS GENERATED
ASSERT BUST (RACH)
ASSERT BUST HOLD (CCBC)
LOCK ASSERTED OR NON-CPU DATIP ?
YES
NO
ASSERT CP CYCLE (CCBB)
ASSERT LOCK (CCBB)
ADDRESS & OPERATION CONTROL BITS GATED INTO CACHE (ADME,F,J)
ADDRESS & OPERATION CONTROL BITS GATED ONTO MAIN MEMORY BUS (ADML)
ADDRESS PROCESSED
MISS (ie, not HIT)
T30
T60
T90
T120
INITIATE CACHE TIMING SEQUENCE (CCBE)
CPU ENTERS PAUSE ROM STATE
ASSERT CONTROL OK (TMCE)
NEGATE BUST HOLD (CCBC)
T150
T150 HOLD
ASSERT START SLOW (CCBD)
ASSERT SLOW CYCLE (CCBD)
DATA DESKEW 100 NS (CCBD)
ASSERT START (CCBD)
LATCH ADDRESS & OPERATION CONTROL LINES
ASSERT MAIN ACK
START READ MEMORY CYCLE
ASSERT MAIN BOCC
36-BIT DOUBLE WORD PLACED ON D-LINES OF MAIN MEMORY BUS
DATA DESKEW
ASSERT MAIN DATA READY
DISABLE TIME OUT (CCBE,D)
LOAD BUS DATA (HI & LO) REGISTER (CDPA)
ASSERT RESTART (CCBE)
ASSERT MEM SYNC SLOW (CCBE)
GENERATE WRITE PULSES (CCBE,DTMB)
ASSERT DONE (CCBC)
ASSERT MEM SYNC (CCBC)
TAG LOADED INTO ADDRESS MEMORY (ADMA-D)
HI & LO WORDS LOADED INTO FDM (DTMB-L)
ONE DATA WORD GATED TO PROCESSOR (CDPB,F,DTMM,P)
NEGATE LOCK (CCBB)
CACHE QUIESCENT
WAIT FOR MEM SYNC (TIGC)
BR LOADED
11-2828

*The processor time states are intended as a frame of reference only for events which occur in the processor.

Figure 3-9 Processor Read Miss

the Cache asserts CP CYCLE and LOCK. LOCK indicates that the Cache is presently "locked" into an operating cycle (CP CYCLE in this case) and that no other requests will be serviced until the present cycle is completed. CP CYCLE causes the address generated by the Memory Management to be gated into the Cache by the Address Multiplexer. This address is processed in the Cache and, at the same time, gated to the Main Memory Bus (along with the control bits), in case a slow cycle to Main Memory will be required. Incoming address bits (9:2) address the FDM to select data to be read if a hit occurs. Bits (9:2) of the incoming address also address the Address Memory. Bits (21:10) of the incoming address are checked against the contents of the Address Memory to determine whether the contents of the address referenced are currently stored in the Cache. HIT 0 or HIT 1 will be asserted if the data being requested is in the FDM. Since this paragraph discusses processor read misses, assume that neither HIT 0 nor HIT 1 is asserted. Assume also that an odd address (XXXXXX2 or XXXXXX6) is being read. Address bit 1 = 1 (odd word address) causes the odd areas of the FDM to be enabled and therefore an odd addressed word is output from each group of the FDM. However, because SLOW CYCLE is asserted during this cycle (see below), the Cache Data Multiplexer ignores the outputs from the FDM.

When LOCK is asserted, a timing sequence is initiated in the Cache. While this sequence is occurring, the processor may abort the read operation by issuing a BEND during T2 of the ROM state following the "BUST" ROM state. However, if the processor does not wish to abort the read operation, it asserts CONTROL OK at T3 of the "PAUSE" ROM state following the "BUST" ROM state.

CONTROL OK, when received by the Cache, allows the generation of START SLOW and the negation of BUST HOLD at T180 of the Cache timing sequence. BUST HOLD negated prevents the Cache from responding twice to the same processor BUST cycle. START SLOW generates SLOW CYCLE and, after a 100 ns deskew delay, START is asserted on the Main Memory Bus. START causes the address and control bits presently on the Main Memory Bus to be loaded into Main Memory. A memory cycle is then started, and MAIN ACK is transmitted from the Main Memory to the Cache. The memory cycle results in two 18-bit words being placed on the data lines of the Main Memory Bus and, after a data deskew delay, the assertion of DATA READY. DATA READY, when received in the Cache, loads the data on the Main Memory Bus into the Bus Data Registers. The outputs of the Bus Data Registers are gated to the FDM and also to the Cache Data Multiplexer. Since we have assumed that the processor is requesting an odd word, ADRS bit 1 is asserted, and causes the Cache Data Multiplexer to select the data stored in the Bus Data (High Word) Register and gate it to the processor.

The receipt of both MAIN ACK and DATA READY in the Cache indicates that the Main Memory has responded properly. Therefore, these signals inhibit the generation of a Main Memory Bus time-out by negating CCBE ALLOW TIMEOUT L. When the time-out is inhibited, write pulses are generated. The write pulses load the two words (block) brought from Main Memory into the FDM and their address tag into the Address Memory. Whether Group 0 or Group 1 of the FDM (and corresponding Tag 0 Address Memory or Tag 1 Address Memory) is loaded is determined as described in Paragraph 4.7. When time-out is inhibited, MEM SYNC SLOW is asserted and causes the assertion of MEM SYNC. MEM SYNC is routed to the processor. Receipt of MEM SYNC allows the processor to proceed past T5 of the PAUSE ROM state, and also causes the data from the Cache to be loaded into the processor's BR. When time-out is inhibited, RESTART is also asserted in the Cache. This asserts DONE, which negates LOCK and brings the Cache to its quiescent state. With LOCK negated, the Cache can begin servicing other requests for memory access.

3.8.3 Processor Write

Figure 3-10 is a flowchart illustrating Cache operation during a processor write cycle. The processor may initiate a data transfer only when it is in a "BUST" ROM state. When performing a write to memory, the processor/Memory Management transmits data gated from the BR, a 22-bit physical address, and operation control bits C1, C0 to the Cache. During the "BUST" ROM state, the processor asserts BUST; this causes BUST HOLD to be asserted in the Cache. A CP cycle will be initiated by the Cache *if*:

1. The Cache is not presently servicing the request of some other device (i.e:, LOCK is not asserted).

2. The Cache is not presently waiting to execute the write portion of a DATIP initiated by some other device.

3. There are no other requests pending (i.e., PRE UBUS or PRE MBC is not asserted).

If the above conditions are satisfied when (or while) BUST is asserted, or even if the above conditions are satisfied when only BUST HOLD is asserted, the Cache asserts CP CYCLE and LOCK. LOCK indicates that the Cache is presently "locked" into an operating cycle (CP CYCLE in this case) and that no other requests will be serviced until the present cycle is completed.

CP CYCLE causes the Cache Write Multiplexer to select the processor data, thereby routing it to the FDM and the high and low word Main Memory Bus Data Drivers. When the Main Memory Bus data lines become free (MAIN BOCC L negated), the Cache enables the data word onto the Main Memory Bus.

CP CYCLE also causes the address generated by the processor and Memory Management to be gated into the Cache by the Address Multiplexer. This address is processed in the Cache and, at the same time, gated to the Main Memory Bus (along with the control bits). Incoming address bits (9:2) address the FDM to select a block in Group 0 and Group 1 which will be updated in case a hit occurs. Bits 0 and 1 of the incoming address select the word or byte in the selected blocks which will be updated in case a hit occurs. Bits (9:2) of the incoming address also address the Address Memory. Bits (21:10) of the incoming address are checked against the contents of the Address Memory to determine whether the contents of the address referenced are currently stored in the Cache. HIT 0 or HIT 1 will be asserted if the data being referenced is in the FDM.

When LOCK is asserted, a timing sequence is initiated in the Cache. While this sequence is occurring, the processor may abort the read operation by issuing a BEND during T2 of the ROM state following the "BUST" ROM state. However, if the processor does not wish to abort the read operation, it asserts CONTROL OK at T3 of the "PAUSE" ROM state following the "BUST" ROM state. CONTROL OK, when received by the Cache, allows the generation of START SLOW and the negation of BUST HOLD at T180 of the Cache timing sequence. BUST HOLD negated prevents the Cache from responding twice to the same processor BUST cycle. START SLOW generates SLOW CYCLE and enables the assertion of START on the Main Memory Bus 100 ns after the bus becomes unoccupied (START WRITE asserted). START causes the address, data, and control bits presently on the Main Memory Bus to be loaded into Main Memory. A memory cycle is then started, and MAIN ACK is transmitted back to the Cache.

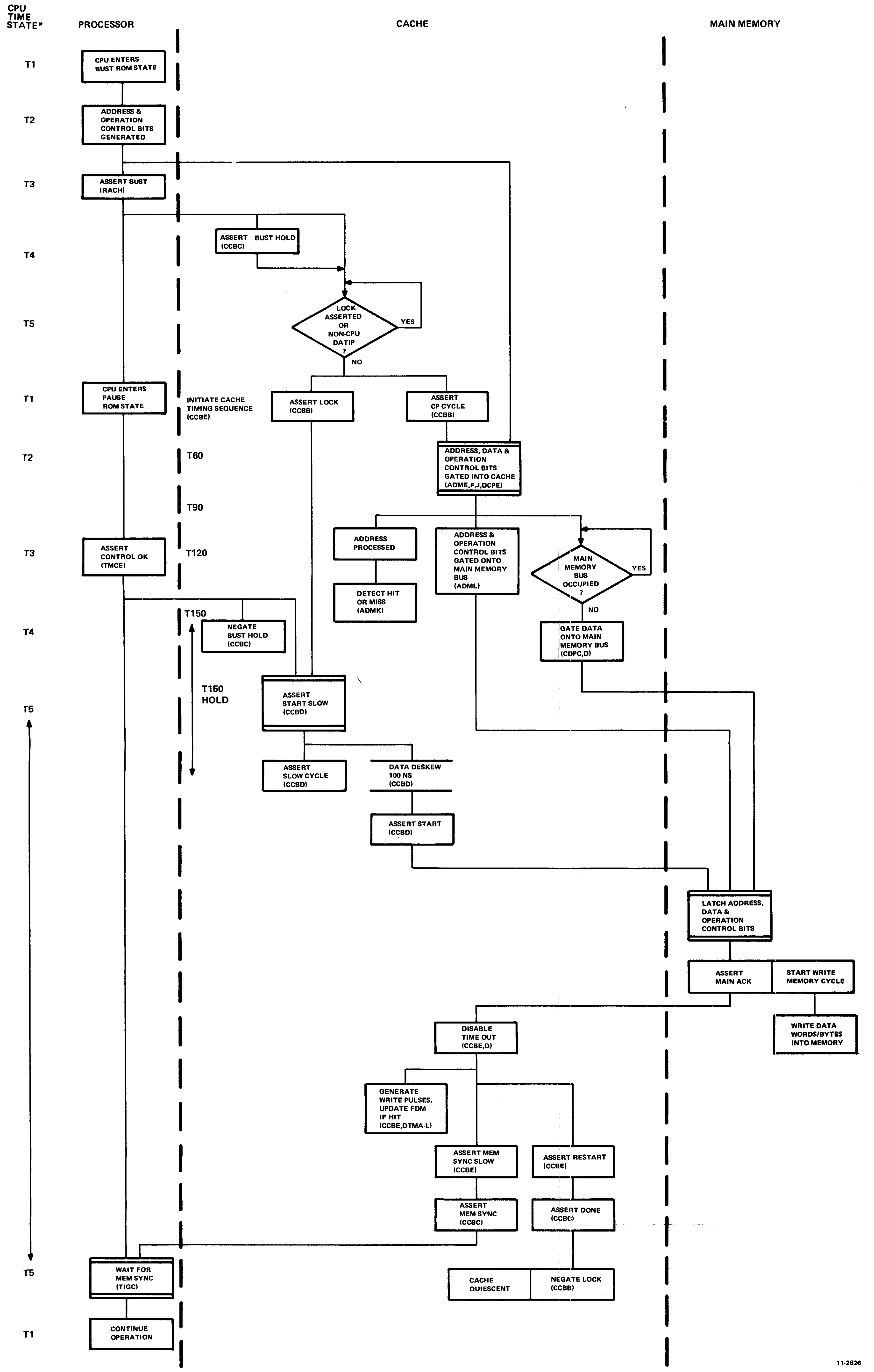
The receipt of MAIN ACK in the Cache indicates that Main Memory has responded properly, and therefore inhibits generation of a Main Memory Bus time-out by negating CCBE ALLOW TIME-OUT L. When the time-out is disabled, write pulses are generated if a hit has been detected. The write pulses (DTMA LO BYTE WP 0*1 L, HI BYTE WP 0*1 L, LO BYTE WP 2*3 L, and/or HI BYTE WP 2*3 L) load the word or byte being written into the FDM group on which the hit occurred at its proper position within the currently indexed block.

When time-out is inhibited, MEM SYNC SLOW is asserted and causes the assertion of MEM SYNC, which is routed to the processor. Receipt of MEM SYNC allows the processor to proceed past T5 of the PAUSE ROM state.

When time-out is inhibited, RESTART is also asserted in the Cache. This asserts DONE, which negates LOCK and brings the Cache to its quiescent state. With LOCK negated, the Cache can begin servicing other requests for memory access.

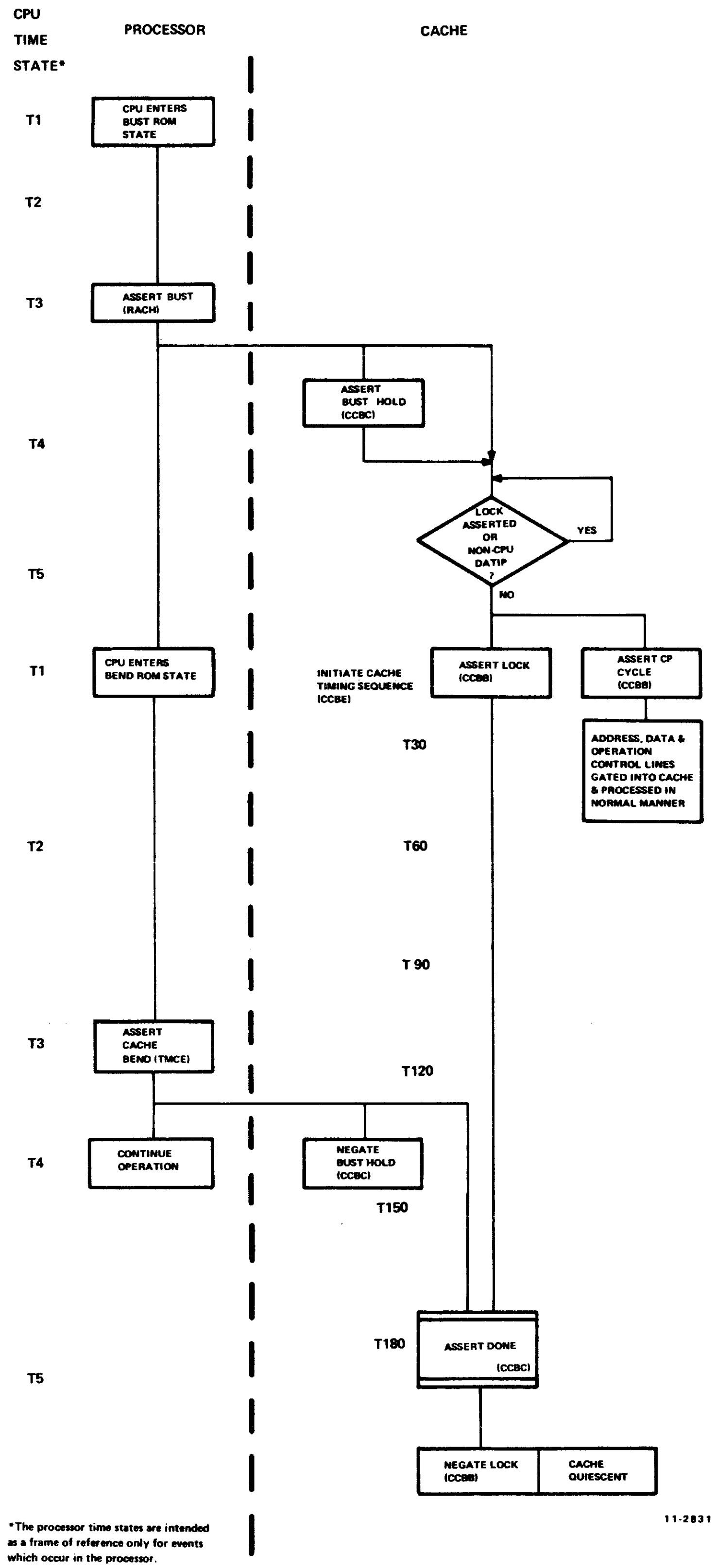
3.8.4 Processor BUST-BEND Cycle

Figure 3-11 is a flowchart illustrating Cache operation during a processor BUST-BEND cycle. The processor often initiates a data transfer (by asserting BUST) that is immediately aborted in the next ROM state (by asserting BEND). This allows the processor to operate more quickly; data transfers can be initiated earlier than otherwise possible and, if they are not required, they are then aborted.


CPU TIME STATE*
PROCESSOR
CACHE
MAIN MEMORY
T1
T2
T3
T4
T5
T1
T2
T3
T4
T5
T5
T1
CPU ENTERS BUST ROM STATE
ADDRESS & OPERATION CONTROL BITS GENERATED
ASSERT BUST (RACH)
ASSERT BUST HOLD (CCBC)
LOCK ASSERTED OR NON-CPU DATIP ?
YES
NO
CPU ENTERS PAUSE ROM STATE
INITIATE CACHE TIMING SEQUENCE (CCBE)
ASSERT LOCK (CCBB)
ASSERT CP CYCLE (CCBB)
T60
ADDRESS, DATA & OPERATION CONTROL BITS GATED INTO CACHE (ADME,F,J,DCPE)
T90
ASSERT CONTROL OK (TMCE)
T120
ADDRESS PROCESSED
ADDRESS & OPERATION CONTROL BITS GATED ONTO MAIN MEMORY BUS (ADML)
MAIN MEMORY BUS OCCUPIED ?
YES
NO
DETECT HIT OR MISS (ADMK)
T150
NEGATE BUST HOLD (CCBC)
GATE DATA ONTO MAIN MEMORY BUS (CDPC,D)
T150 HOLD
ASSERT START SLOW (CCBD)
ASSERT SLOW CYCLE (CCBD)
DATA DESKEW 100 NS (CCBD)
ASSERT START (CCBD)
LATCH ADDRESS, DATA & OPERATION CONTROL BITS
ASSERT MAIN ACK
START WRITE MEMORY CYCLE
WRITE DATA WORDS/BYTES INTO MEMORY
DISABLE TIME OUT (CCBE,D)
GENERATE WRITE PULSES. UPDATE FDM IF HIT (CCBE,DTMA-L)
ASSERT MEM SYNC SLOW (CCBE)
ASSERT RESTART (CCBE)
ASSERT MEM SYNC (CCBC)
ASSERT DONE (CCBC)
WAIT FOR MEM SYNC (TIGC)
CACHE QUIESCENT
NEGATE LOCK (CCBB)
CONTINUE OPERATION
11-2826

*The processor time states are intended as a frame of reference only for events which occur in the processor.

Figure 3-10 Processor Write



*The processor time states are intended as a frame of reference only for events which occur in the processor.

Figure 3-11 Processor Bust-Bend Cycle

Cache timing during a BUST-BEND cycle is similar to that of a processor read hit, as illustrated in Figure 3-8. During the "BUST" ROM state, the processor asserts BUST; this causes BUST HOLD to be asserted in the Cache. A CP cycle will be initiated by the Cache *if*:

1. The Cache is not presently servicing the request of some other device.

2. The Cache is not presently waiting to execute the write portion of a DATIP initiated by some other device.

3. There are no other requests pending (i.e., PRE UBUS or PRE MBC are not asserted).

If the above conditions are satisfied when (or while) BUST is asserted, or even if the above conditions are satisfied when only BUST HOLD is asserted, the Cache asserts CP CYCLE and LOCK. (The processor could assert BEND prior to the assertion of LOCK, in which case BUST HOLD is immediately negated and the cycle is aborted before the Cache actually begins executing the cycle.) LOCK indicates that the Cache is presently "locked" into an operating cycle (CP CYCLE in this case) and that no other requests will be serviced until the present cycle is completed. CP CYCLE causes the address, control, and data bits currently being output from the processor/Memory Management to be gated and processed in the Cache, in the same manner they normally would be for a read or write operation.

When LOCK is asserted, a timing sequence is initiated in the Cache. While this sequence is occurring, the processor may abort the operation by issuing a BEND during T2 of the ROM state following the BUST ROM state. When it does so, BUST HOLD is negated in the Cache and BEND HOLD is asserted. BEND HOLD asserted causes the assertion of DONE at T180 of the Cache timing sequence. This negates LOCK and thereby brings the Cache to its quiescent state. With LOCK negated, the Cache can begin servicing other requests for memory access.

Note that during a BUST-BEND cycle, the processor does not issue CONTROL OK. This prevents the Cache from starting a slow cycle or asserting MEM SYNC.

3.8.5 Unibus Map Read Hit

Figure 3-12 is a flowchart illustrating Cache operation during a Unibus Map read hit cycle. When performing a read operation via the Cache, the Unibus Map transmits a 22-bit address (generated from the 18-bit Unibus address) and Control bits C1, C0 to the Cache, and asserts UB REQUEST. UB REQUEST, delayed and synchronized by a Cache internal clock (SYNC CLK), generates PRE UBUS (Pre Unibus Cycle), which will initiate Cache operation to service the Unibus request *if*:

1. The Cache is not presently servicing the request of some other device.

2. The Cache is not waiting to execute the write portion of a DATIP initiated by some other device.

If the above conditions are satisfied, the Cache asserts UB CYCLE and LOCK. LOCK indicates that the Cache is presently "locked" into an operating cycle (UB CYCLE in this case) and that no other requests will be serviced until the present cycle is completed. UB CYCLE causes the address generated by the Unibus Map to be gated into the Cache by the Address Multiplexer. This address is processed in the Cache and, at the same time, gated to the Main Memory Bus (along with control bits), in case a slow cycle to Main Memory will be required. Incoming address bits (9:2) address the FDM to select data to be read in case a hit occurs. Bits (9:2) of the incoming address also address the Address Memory. Bits (21:10) of the incoming address are checked against the contents of the Address Memory to determine whether the contents of the address referenced are currently stored in the Cache. HIT 0 or HIT 1 will be asserted if the data being requested is in the FDM. Since this paragraph discusses Unibus Map read hits, assume that HIT 0 is asserted and that an odd address (XXXXXX2 or XXXXXX6) is being read. Address bit 1 = 1 (odd address) causes the odd areas of the FDM to be enabled and therefore an odd addressed word is output from each group of the FDM. HIT 0 asserted causes the Cache Data Multiplexer to gate out only the odd addressed word from Group 0 of the FDM; this word is routed to the Unibus Map.

When LOCK is asserted, a timing sequence is initiated in the Cache. At T30, UB ACKN is asserted and transmitted to the Unibus Map. This signal negates the Unibus Map request, thereby preventing

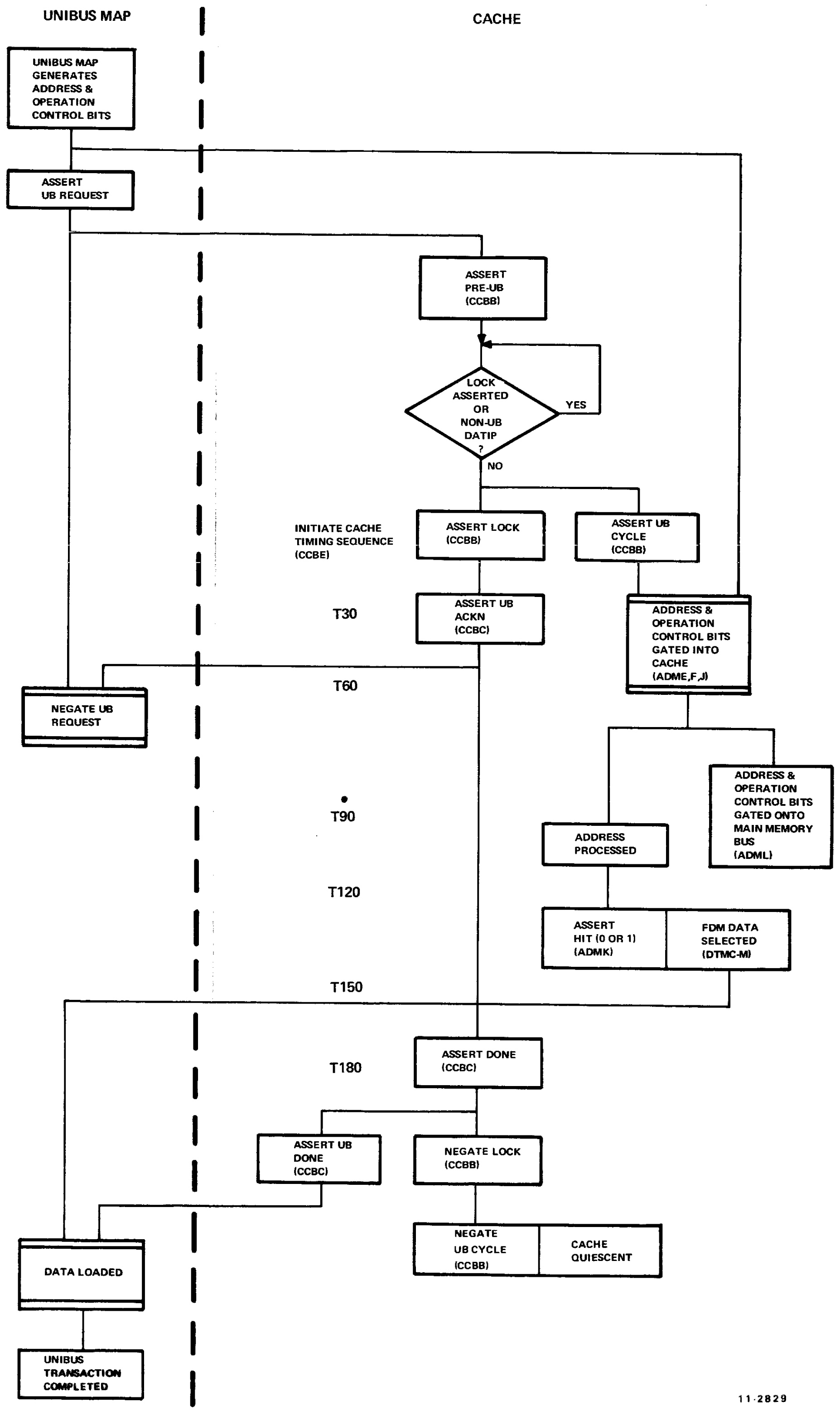

UNIBUS MAP
CACHE
UNIBUS MAP GENERATES ADDRESS & OPERATION CONTROL BITS
ASSERT UB REQUEST
ASSERT PRE-UB (CCBB)
LOCK ASSERTED OR NON-UB DATIP ?
YES
NO
INITIATE CACHE TIMING SEQUENCE (CCBE)
ASSERT LOCK (CCBB)
ASSERT UB CYCLE (CCBB)
T30
ASSERT UB ACKN (CCBC)
ADDRESS & OPERATION CONTROL BITS GATED INTO CACHE (ADME,F,J)
T60
NEGATE UB REQUEST
ADDRESS & OPERATION CONTROL BITS GATED ONTO MAIN MEMORY BUS (ADML)
T90
ADDRESS PROCESSED
T120
ASSERT HIT (0 OR 1) (ADMK)
FDM DATA SELECTED (DTMC-M)
T150
T180
ASSERT DONE (CCBC)
ASSERT UB DONE (CCBC)
NEGATE LOCK (CCBB)
DATA LOADED
NEGATE UB CYCLE (CCBB)
CACHE QUIESCENT
UNIBUS TRANSACTION COMPLETED
11-2829

Figure 3-12 Unibus Map Read Hit

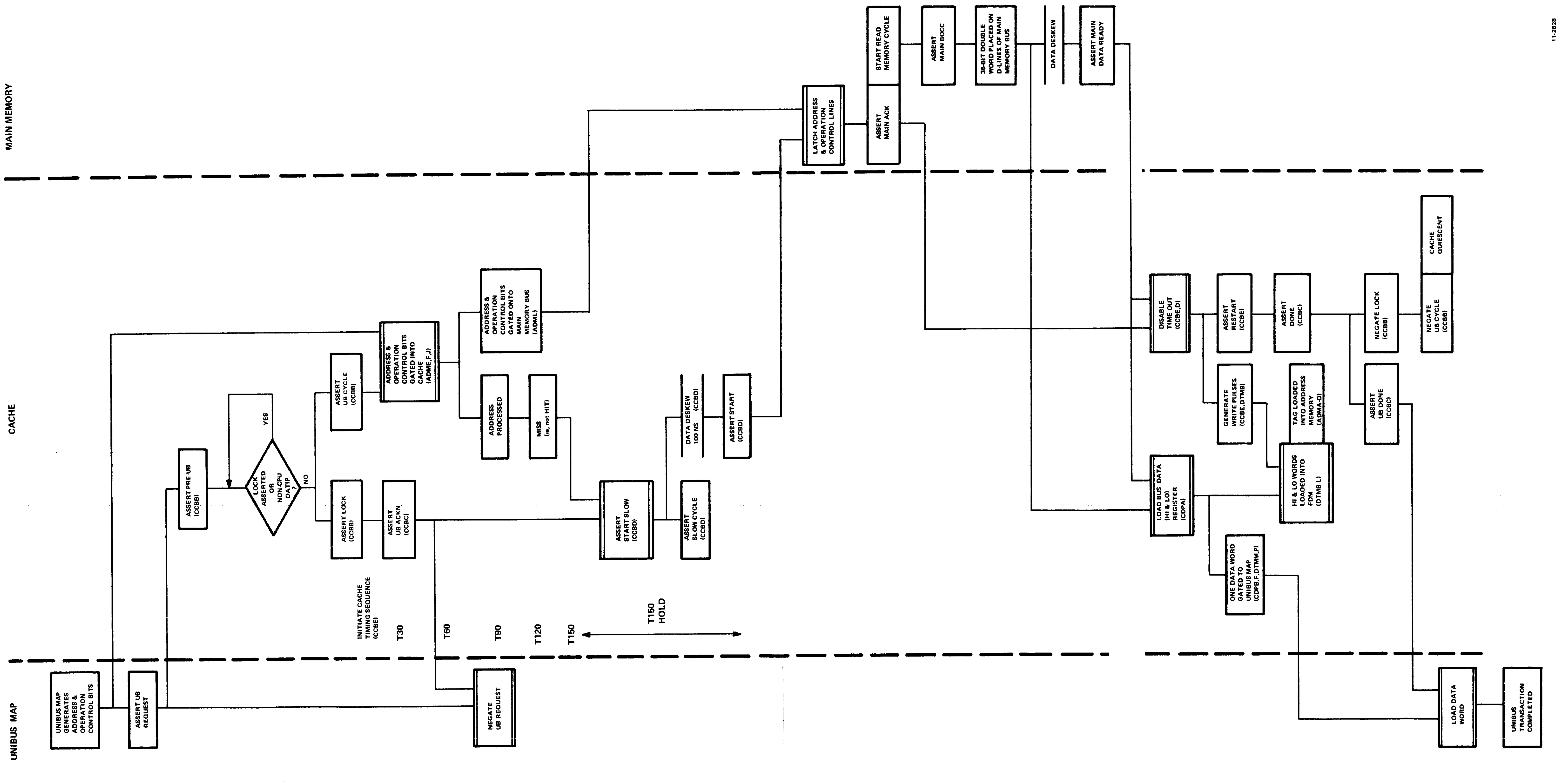

MAIN MEMORY
CACHE
UNIBUS MAP
UNIBUS MAP GENERATES ADDRESS & OPERATION CONTROL BITS
ASSERT UB REQUEST
ASSERT PRE-UB (CCBB)
LOCK ASSERTED OR NON-CPU DATIP ?
YES
NO
ASSERT UB CYCLE (CCBB)
ASSERT LOCK (CCBB)
ASSERT UB ACKN (CCBC)
ADDRESS & OPERATION CONTROL BITS GATED INTO CACHE (ADME,F,J)
ADDRESS & OPERATION CONTROL BITS GATED ONTO MAIN MEMORY BUS (ADML)
ADDRESS PROCESSED
MISS (ie, not HIT)
ASSERT START SLOW (CCBD)
ASSERT SLOW CYCLE (CCBD)
DATA DESKEW 100 NS (CCBD)
ASSERT START (CCBD)
INITIATE CACHE TIMING SEQUENCE (CCBE)
T30
T60
T90
T120
T150
T150 HOLD
NEGATE UB REQUEST
LATCH ADDRESS & OPERATION CONTROL LINES
ASSERT MAIN ACK
START READ MEMORY CYCLE
ASSERT MAIN BOCC
36-BIT DOUBLE WORD PLACED ON D-LINES OF MAIN MEMORY BUS
DATA DESKEW
ASSERT MAIN DATA READY
DISABLE TIME OUT (CCBE,D)
LOAD BUS DATA (HI & LO) REGISTER (CDPA)
ASSERT RESTART (CCBE)
GENERATE WRITE PULSES (CCBE,DTMB)
ASSERT DONE (CCBC)
TAG LOADED INTO ADDRESS MEMORY (ADMA-D)
HI & LO WORDS LOADED INTO FDM (DTMB-L)
ONE DATA WORD GATED TO UNIBUS MAP (CDPB,F,DTMM,P)
NEGATE LOCK (CCBB)
ASSERT UB DONE (CCBC)
CACHE QUIESCENT
NEGATE UB CYCLE (CCBB)
LOAD DATA WORD
UNIBUS TRANSACTION COMPLETED
11-2828

Figure 3-13 Unibus Map Read Miss

the same request from being serviced twice. At T180, DONE is asserted in the Cache. This signal generates UB DONE, which loads the FDM data gated out of the Cache into the Unibus Map, and causes the Unibus Map to terminate its transaction on the Unibus by issuing SSYN. DONE also brings the Cache into its quiescent state by negating LOCK, which in turn negates UB CYCLE. With LOCK negated, the Cache can begin servicing other requests for memory access.

3.8.6 Unibus Map Read Miss

Figure 3-13 is a flowchart illustrating Cache operation during a Unibus Map read miss cycle. When performing a read operation via the Cache, the Unibus Map transmits a 22-bit address (generated from the 18-bit Unibus address) and control bits C1, C0 to the Cache, and asserts UB REQUEST. UB REQUEST, delayed and synchronized by a Cache internal clock (SYNC CLK), generates PRE UBUS (Pre Unibus Cycle), which will initiate Cache operation to service the Unibus request *if*:

1. The Cache is not presently servicing the request of some other device (i.e., LOCK not asserted).

2. The Cache is not waiting to execute the write portion of a DATIP initiated by some other device.

If the above conditions are satisfied, the Cache asserts UB CYCLE and LOCK. LOCK indicates that the Cache is presently "locked" into an operating cycle (UB CYCLE in this case) and that no other requests will be serviced until the present cycle is completed. UB CYCLE causes the address generated by the Unibus Map to be gated into the Cache by the Address Multiplexer. This address is processed in the Cache and, at the same time, gated to the Main Memory Bus (along with the control bits), in case a slow cycle to Main Memory will be required. Incoming address bits (9:2) address the FDM to select data to be read in case a hit occurs. Bits (9:2) of the incoming address also address the Address Memory. Bits (21:10) of the incoming address are checked against the contents of the Address Memory to determine whether the contents of the address referenced are currently stored in the Cache. Since this paragraph discusses Unibus Map read misses, assume that neither HIT 0 nor HIT 1 is asserted. Assume also that an odd address (XXXXXX2 or XXXXXX6) is being read. Address bit 1 = 1 (odd address) causes the odd areas of the FDM to be enabled and therefore an odd addressed word is output from each group of the FDM. However, because SLOW CYCLE is asserted during this cycle (see below), the Cache Data Multiplexer ignores the outputs from the FDM.

When LOCK is asserted, a timing sequence is initiated in the Cache. At T30, UB ACKN is asserted and transmitted to the Unibus Map. This signal negates the Unibus Map request, thereby preventing the same request from being serviced twice. At T180, START SLOW is asserted in the Cache. This asserts SLOW CYCLE and, after a 100 ns skew delay, START is asserted on the Main Memory Bus. START causes the address and control bits presently on the Main Memory Bus to be loaded into Main Memory. A memory cycle is then started, and MAIN ACK is transmitted back to the Cache. The memory cycle results in two 18-bit words being placed on the data lines of the Main Memory Bus and, after a data deskew delay, the assertion of DATA READY. DATA READY, when received in the Cache, loads the data on the Main Memory Bus into the Bus Data Registers. The outputs of the Bus Data Registers are gated to the FDM and also to the Cache Data Multiplexer. Since we have assumed that the Unibus Map is requesting an odd word, ADRS bit 1 is asserted, and causes the Cache Data Multiplexer to select the data stored in the Bus Data (High Word) Register and gate it to the Unibus Map.

The receipt of both MAIN ACK and DATA READY in the Cache indicates that the Main Memory has responded properly. Therefore, these signals inhibit the generation of a Main Memory Bus time-out by negating CCBE ALLOW TIMEOUT L. When the time-out is inhibited, write pulses are generated. The write pulses load the two 18-bit words (i.e., the block) brought from Main Memory into the FDM and their identification bits into the Address Memory. Whether Group 0 or Group 1 of the FDM (and corresponding Tag 0 Address Memory or Tag 1 Address Memory) is loaded is determined as described in Paragraph 4.7.

When time-out is inhibited, RESTART is asserted, which in turn asserts DONE. DONE generates UB DONE, which loads the data word gated out of the Cache into the Unibus Map. The Unibus Map will place the data word on the Unibus and then complete its Unibus transaction. DONE also brings the Cache into its quiescent state by negating LOCK, which in turn negates UB CYCLE. With LOCK negated, the Cache can begin servicing other requests for memory access.

3.8.7 Unibus Map Write

Figure 3-14 is a flowchart illustrating Cache operation during a Unibus Map write cycle. When performing a write to memory, the Unibus Map transmits data gated from the Unibus, a 22-bit physical address (generated from the 18-bit Unibus Address), and operation control bits C1 and C0 to the Cache, and then asserts UB REQUEST. UB REQUEST, delayed and synchronized by a Cache internal clock (SYNC CLK), generates PRE UBUS (Pre Unibus Cycle), which will initiate Cache operation to service the Unibus request *if*:

1. The Cache is not presently servicing the request of some other device (i.e., LOCK not asserted).

2. The Cache is not waiting to execute the write portion of a DATIP initiated by some other device.

If the above conditions are satisfied, the Cache asserts UB CYCLE and LOCK. LOCK indicates that the Cache is presently "locked" into an operating cycle (UB CYCLE in this case) and that no other requests will be serviced until the present cycle is completed.

UB CYCLE causes the Cache Write Multiplexer to select the Unibus data, thereby routing it to the FDM and the (high and low word) Main Memory Bus Data Drivers. When the Main Memory Bus data lines become free (MAIN BOCC L negated), the Cache enables the data word onto the Main Memory Bus.

UB CYCLE also causes the address generated by the Unibus Map to be gated into the Cache by the Address Multiplexer. This address is processed in the Cache and, at the same time, gated to the Main Memory Bus (along with control bits). Incoming address bits (9:2) address the FDM to select a block in Group 0 and Group 1 which will be updated in case a hit occurs. Bits 1 and 0 of the incoming address select a word or byte in the selected blocks which will be updated in case a hit occurs. Bits (9:2) of the incoming address also address the Address Memory. Bits (21:10) of the incoming address are checked against the contents of the Address Memory to determine whether the contents of the address referenced are currently stored in the Cache. HIT 0 or HIT 1 will be asserted if the data being referenced is in the FDM.

When LOCK is asserted, a timing sequence is initiated in the Cache. At T30, UB ACKN is asserted and transmitted to the Unibus Map. This signal negates the Unibus Map request, thereby preventing the same request from being serviced twice. At T180, START SLOW is asserted in the Cache. This asserts SLOW CYCLE and enables the assertion of START on the Main Memory Bus 100 ns after the bus becomes unoccupied (START WRITE asserted). START causes the address, data, and control bits presently on the Main Memory Bus to be loaded into Main Memory. A memory cycle is then started, and MAIN ACK is transmitted back to the Cache.

The receipt of MAIN ACK in the Cache indicates that Main Memory has responded properly, and therefore inhibits generatioh of a Main Memory Bus time-out. When the time-out is inhibited, write pulses are generated if a hit has been detected. The write pulses (DTMB LO BYTE WP 0*1 L, HI BYTE WP 0*1 L, LO BYTE WP 2*3 L, and/or HI BYTE WP 2*3 L) load the word or byte being written into the FDM group on which the hit occurred at its proper position within the currently indexed block. When time-out is inhibited, RESTART is asserted, which in turn causes the assertion of DONE; DONE generates UB DONE, which is transmitted to the Unibus Map and informs it that the write operation has been executed; this allows the Unibus Map to terminate its transaction on the Unibus by issuing SSYN. DONE also brings the Cache into its quiescent state by negating LOCK, which in turn negates UB CYCLE. With LOCK negated, the Cache can begin servicing other requests for memory access.

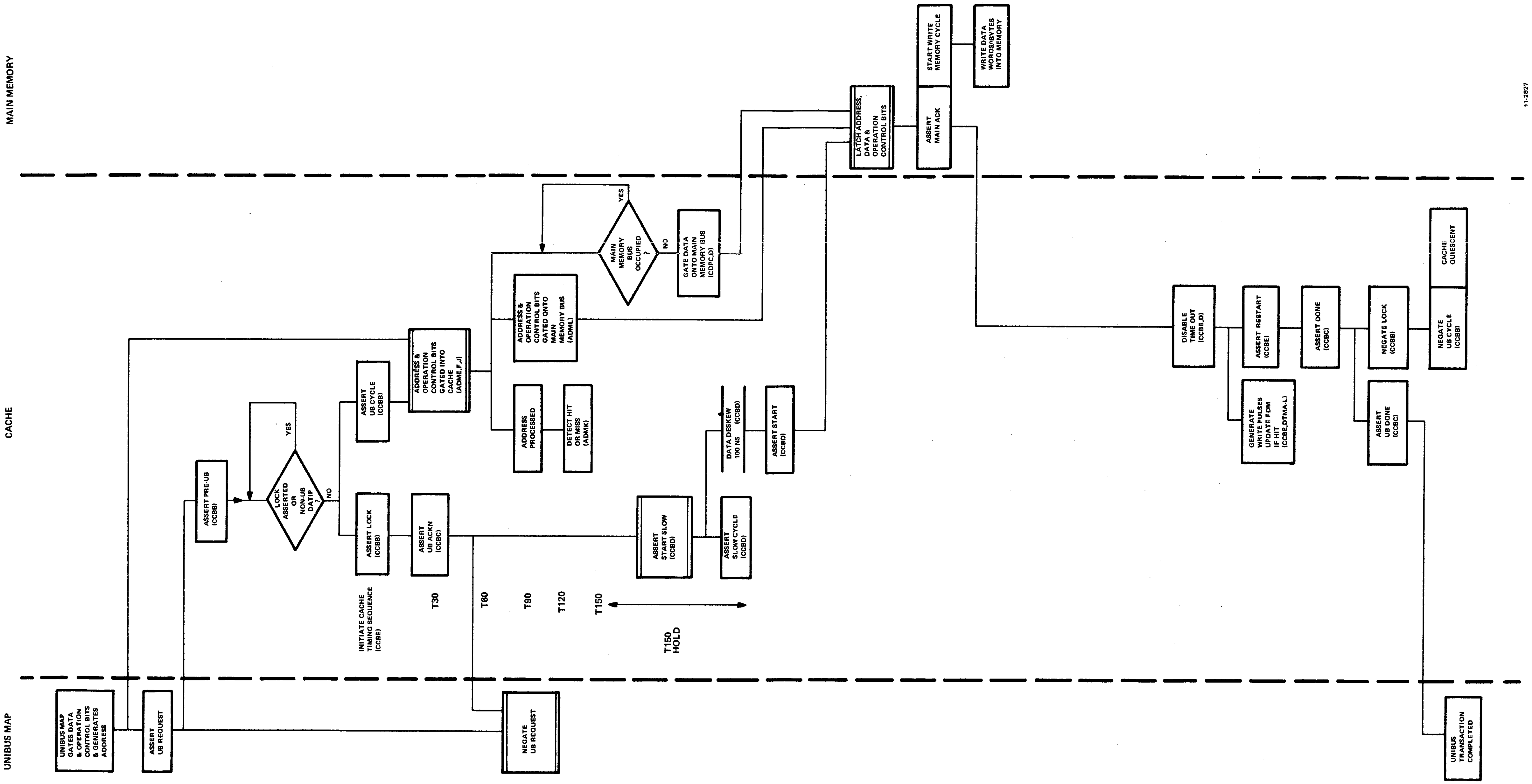

UNIBUS MAP
CACHE
MAIN MEMORY
UNIBUS MAP GATES DATA & OPERATION CONTROL BITS & GENERATES ADDRESS
ASSERT UB REQUEST
ASSERT PRE-UB (CCBB)
LOCK ASSERTED OR NON-UB DATIP ?
YES
NO
ASSERT UB CYCLE (CCBB)
ASSERT LOCK (CCBB)
ASSERT UB ACKN (CCBC)
INITIATE CACHE TIMING SEQUENCE (CCBE)
T30
T60
T90
T120
T150
T150 HOLD
NEGATE UB REQUEST
ADDRESS & OPERATION CONTROL BITS GATED INTO CACHE (ADME,F,J)
ADDRESS & OPERATION CONTROL BITS GATED ONTO MAIN MEMORY BUS (ADML)
ADDRESS PROCESSED
DETECT HIT OR MISS (ADMK)
MAIN MEMORY BUS OCCUPIED ?
YES
NO
GATE DATA ONTO MAIN MEMORY BUS (CDPC,D)
ASSERT START SLOW (CCBD)
ASSERT SLOW CYCLE (CCBD)
DATA DESKEW 100 NS (CCBD)
ASSERT START (CCBD)
LATCH ADDRESS, DATA & OPERATION CONTROL BITS
ASSERT MAIN ACK
START WRITE MEMORY CYCLE
WRITE DATA WORDS/BYTES INTO MEMORY
11-2827
DISABLE TIME OUT (CCBE,D)
ASSERT RESTART (CCBE)
GENERATE WRITE PULSES UPDATE FDM IF HIT (CCBE,DTMA-L)
ASSERT DONE (CCBC)
NEGATE LOCK (CCBB)
ASSERT UB DONE (CCBC)
NEGATE UB CYCLE (CCBB)
CACHE QUIESCENT
UNIBUS TRANSACTION COMPLETED

Figure 3-14 Unibus Map Write

3.8.8 Cache Register Read/Write

Figure 3-15 is a flowchart illustrating Cache operation during a Cache register read or write. A Cache register read or write operation is quite similar to a Unibus Map read hit. When the Unibus Map decodes a Cache register address on the Unibus, it gates bits (03:01) (MAPA ADRS 03:01) of the Unibus address and transmits MAPB CACHE REG L to the Cache. When the Unibus Map receives MSYN on the Unibus, it asserts UB REQUEST. UB REQUEST, delayed and synchronized by a Cache internal clock (SYNC CLK) generates PRE UBUS (Pre Unibus Cycle), which will initiate Cache operation to service the Unibus request *if*:

1. The Cache is not presently servicing the request of some other device.
2. The Cache is not waiting to execute the write portion of a DATIP initiated by some other device.

If the above conditions are satisfied, the Cache asserts UB CYCLE and LOCK. LOCK indicates that the Cache is presently "locked" into an operating cycle (UB CYCLE in this case) and that no other requests will be serviced until the present cycle is completed. UB CYCLE causes the 22 physical address bits and the operation control bits (C1 and C0) to be gated into the Cache by the Address Multiplexer. The 22-bit physical address gated out of the Unibus during a Cache register operation is not a valid address. It is gated into the Cache, indexes into the FDM and Address Memory, and may cause HIT 0 or HIT 1 to be asserted; however, because MAPB CACHE REG L is asserted, assertion of CCBD START SLOW (1) H is inhibited; therefore, reading or writing into the FDM or Main Memory is also inhibited. Address bits MAPA ADRS (03:01) select the desired Cache register. The selected register data is gated out of the Cache to the Unibus Map on the CCB REG D 15:00 lines (whether or not a read or a write is being performed).

When LOCK is asserted, a timing sequence is initiated in the Cache. At T30, UB ACKN is asserted and transmitted to the Unibus Map. This signal negates the Unibus Map request, thereby preventing the same request from being serviced twice.

If a register write operation is being performed, a register write pulse is generated (CCBH WRITE ERR REG L, CCBH CLK MAINT REG L, or CCBH CLK CONTROL REG L) at T60 of the Cache timing sequence.

At T180, DONE is asserted in the Cache. This signal generates UB DONE, which enables the Unibus Map to accept register data (if it is performing a read) and to terminate its transaction on the Unibus. DONE also brings the Cache into its quiescent state by negating LOCK, which in turn negates UB CYCLE. With LOCK negated, the Cache can begin servicing other requests for memory access.

3.8.9 MBC Read From Memory

The flowchart in Figure 3-16 shows a single MBC (MBC A) requesting memory access. If two or more MBCs request memory access concurrently, Cache operation is similar. Multiple MBC requests are discussed in Paragraph 4.6, which describes the Massbus arbitrator.

MBCs requesting memory access assert their respective request signals [CSTC CTRLA (B, C, or D) REQ L]. The Cache MBC arbitrator receives these requests and arbitrates among the requesting MBCs. The MBC arbitration logic asserts MBC REQ and transmits SEL ADRS CTRL "X" H to the selected MBC (MBC A in this case). Receipt of the SEL ADRS signal enables MBC A to gate out an address and operation control lines C1, C0, and CX to the Cache. MBC REQ, delayed and synchronized by a Cache internal clock (SYNC CLK), generates PRE MBC (Pre MBC Cycle), which causes the address and control bits gated out by the selected MBC to be loaded into the MBC Address Latch. PRE MBC will initiate Cache operation to service the MBC request *if*:

1. The Cache is not presently servicing the request of some other device (i.e., LOCK not asserted).
2. The Cache is not waiting to execute the write portion of a DATIP initiated by some other device.
3. There are no Unibus Map requests currently pending (i.e., PRE UBUS is not asserted).

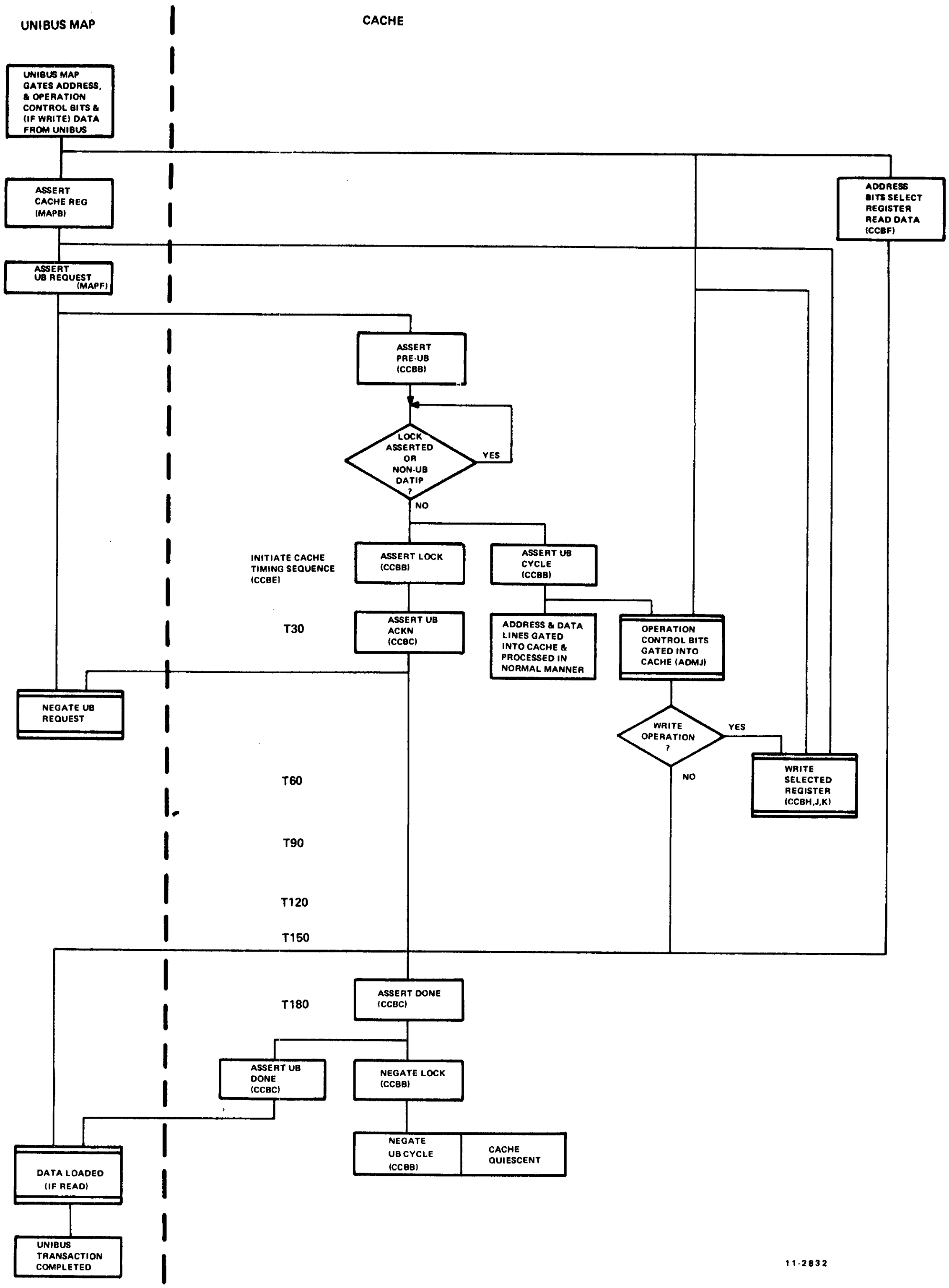

UNIBUS MAP
CACHE
UNIBUS MAP GATES ADDRESS, & OPERATION CONTROL BITS & (IF WRITE) DATA FROM UNIBUS
ASSERT CACHE REG (MAPB)
ADDRESS BITS SELECT REGISTER READ DATA (CCBF)
ASSERT UB REQUEST (MAPF)
ASSERT PRE-UB (CCBB)
LOCK ASSERTED OR NON-UB DATIP ?
YES
NO
INITIATE CACHE TIMING SEQUENCE (CCBE)
ASSERT LOCK (CCBB)
ASSERT UB CYCLE (CCBB)
T30
ASSERT UB ACKN (CCBC)
ADDRESS & DATA LINES GATED INTO CACHE & PROCESSED IN NORMAL MANNER
OPERATION CONTROL BITS GATED INTO CACHE (ADMJ)
NEGATE UB REQUEST
WRITE OPERATION ?
YES
NO
WRITE SELECTED REGISTER (CCBH,J,K)
T60
T90
T120
T150
T180
ASSERT DONE (CCBC)
ASSERT UB DONE (CCBC)
NEGATE LOCK (CCBB)
NEGATE UB CYCLE (CCBB)
CACHE QUIESCENT
DATA LOADED (IF READ)
UNIBUS TRANSACTION COMPLETED
11-2832

Figure 3-15 Register Read and Write

If the above conditions are satisfied, the Cache asserts MBC CYCLE and LOCK. LOCK indicates that the Cache is presently "locked" into an operating cycle (MBC CYCLE in this case) and that no other requests will be serviced until the present cycle is completed. MBC CYCLE causes the address generated by the selected MBC to be gated into the Cache by the Address Multiplexer. This address is processed in the Cache and, at the same time, gated to the Main Memory Bus along with control bits C1 and C0. Incoming address bits (9:2) address the FDM and the Address Memory. Bits (21:10) of the incoming address are checked against the contents of the Address Memory to determine whether the contents of the address referenced are currently stored in the Cache. HIT 0 or HIT 1 will be asserted if the data being requested is in the FDM; however, since data is not read from the FDM during MBC cycles, these signals are not used.

When LOCK is asserted, a timing sequence is initiated in the Cache. At T30 of the timing sequence, DISABLE REQ is asserted. DISABLE REQ clocks the MBC arbitration logic and causes SEL DATA CTRL A to be transmitted to the selected MBC; the MBC ignores this signal during a read from memory. While DISABLE REQ is asserted, the MBC arbitrator is prevented from arbitrating new incoming requests.

At T60, MBC REQ ACKN is asserted. MBC REQ ACKN is transmitted to all the MBCs and notifies the selected MBC that it can negate its request and alter the address and operation control bits.

At T120 of the Cache timing sequence, CLK PRI H (Clock Priority) is generated, and clocks the MBC priority arbitration logic; this records that MBC "A" is currently selected and will influence future selections.

At T150 of the Cache timing sequence, DISABLE REQ is negated. DISABLE REQ negated enables clocking the MBC priority arbitration logic, causing selection of the next MBC if an MBC request is pending.

At T180, START SLOW is asserted in the Cache. This asserts SLOW CYCLE and, after a 100 ns skew delay, START is asserted on the Memory Bus. START causes the address and control bits presently on the Main Memory Bus to be loaded into Main Memory. A memory cycle is then started, and MAIN ACK is transmitted back to the Cache.

The receipt of MAIN ACK in the Cache indicates that the Main Memory is responding properly; therefore, this signal inhibits the generation of a Main Memory Bus time-out by negating CCBE ALLOW TIMEOUT L. Also in response to MAIN ACK, the Cache transmits ADRS ACKN to the MBCs. Time-out inhibited causes the assertion of RESTART, and this in turn causes the assertion of DONE. DONE brings the Cache into its quiescent state by negating LOCK. With LOCK negated, the Cache can begin servicing other requests for memory access.

The memory cycle just initiated results in two 18-bit words being placed on the data lines of the Main Memory Bus and, after a data deskew delay, the assertion of DATA READY. The Main Memory Bus data is received in the Cache and routed to the MBCs. The MBC Arbitration Logic in the Cache routes DATA READY to the MBC performing the read operation (MBC A in this case) by asserting CDPK DATA RDY CNTL A H. This causes MBC A to accept the read data. Note that while the Cache is routing the MBC data and DATA RDY signals, it may already be in the midst of servicing some other request for memory access.

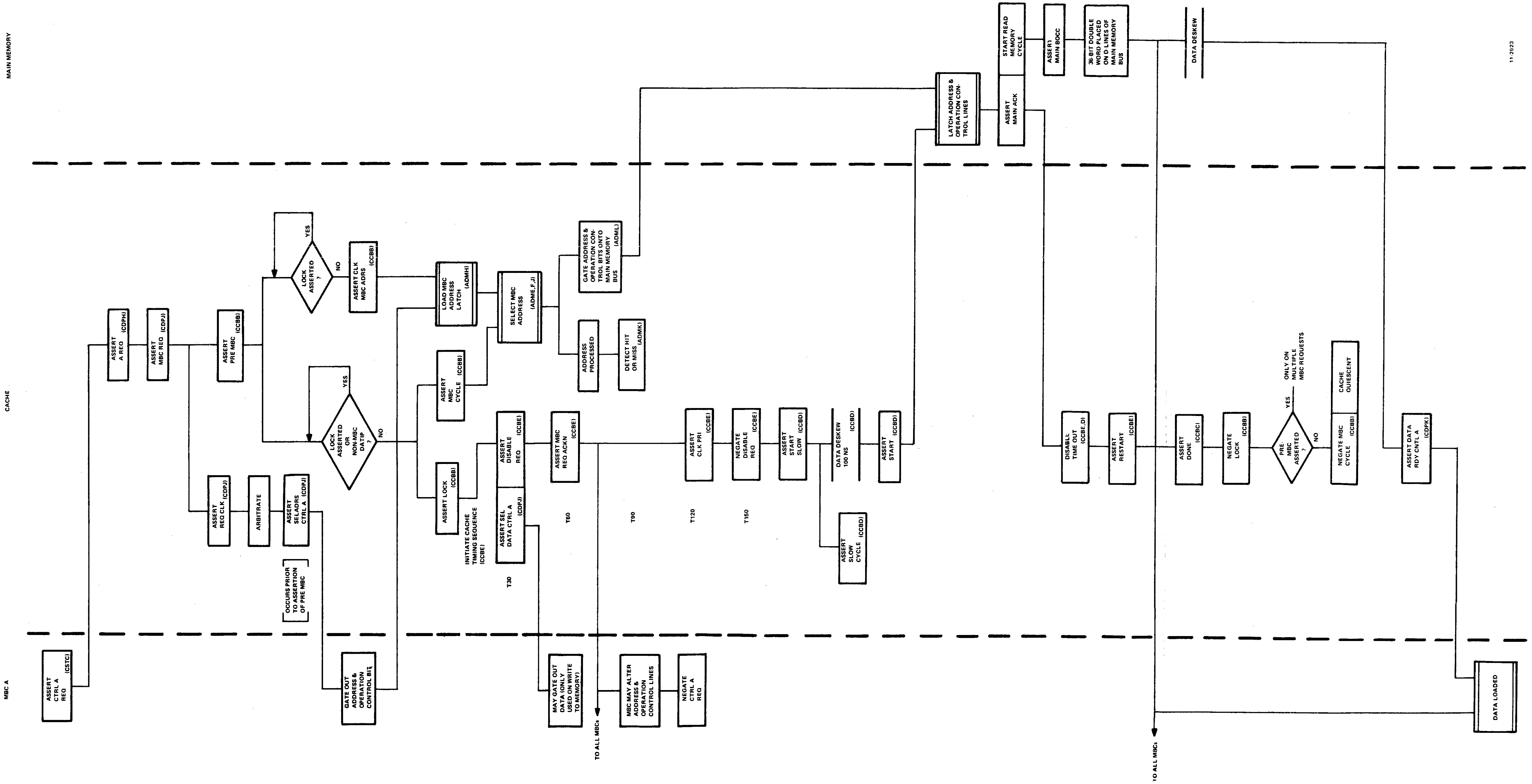

MAIN MEMORY
CACHE
MBC A
ASSERT CTRL A REQ (CSTC)
ASSERT A REQ (CDPH)
ASSERT MBC REQ (CDPJ)
ASSERT PRE MBC (CCBB)
ASSERT REQ CLK (CDPJ)
ARBITRATE
ASSERT SELADRS CTRL A (CDPJ)
OCCURS PRIOR TO ASSERTION OF PRE MBC
GATE OUT ADDRESS & OPERATION CONTROL BIT
LOCK ASSERTED ?
YES
NO
ASSERT CLK MBC ADRS (CCBB)
LOCK ASSERTED OR NON-MBC DATIP ?
YES
NO
LOAD MBC ADDRESS LATCH (ADMH)
ASSERT MBC CYCLE (CCBB)
ASSERT LOCK (CCBB)
SELECT MBC ADDRESS (ADME,F,J)
GATE ADDRESS & OPERATION CONTROL BITS ONTO MAIN MEMORY BUS (ADML)
ADDRESS PROCESSED
DETECT HIT OR MISS (ADMK)
INITIATE CACHE TIMING SEQUENCE (CCBE)
ASSERT DISABLE REQ (CCBE)
ASSERT SEL DATA CTRL A (CDPJ)
T30
ASSERT MBC REQ ACKN (CCBE)
T60
T90
ASSERT CLK PRI (CCBE)
T120
NEGATE DISABLE REQ (CCBE)
T150
ASSERT START SLOW (CCBD)
DATA DESKEW 100 NS (CCBD)
ASSERT SLOW CYCLE (CCBD)
ASSERT START (CCBD)
MAY GATE OUT DATA (ONLY USED ON WRITE TO MEMORY)
TO ALL MBCs
MBC MAY ALTER ADDRESS & OPERATION CONTROL LINES
NEGATE CTRL A REQ
LATCH ADDRESS & OPERATION CONTROL LINES
ASSERT MAIN ACK
START READ MEMORY CYCLE
ASSERT MAIN BOCC
36-BIT DOUBLE WORD PLACED ON D LINES OF MAIN MEMORY BUS
DATA DESKEW
DISABLE TIME OUT (CCBE,D)
ASSERT RESTART (CCBE)
ASSERT DONE (CCRC)
NEGATE LOCK (CCBB)
PRE MBC ASSERTED ?
YES
ONLY ON MULTIPLE MBC REQUESTS
NO
CACHE QUIESCENT
NEGATE MBC CYCLE (CCBB)
ASSERT DATA RDY CNTL A (CDPK)
DATA LOADED
TO ALL MBCs
11-2023

Figure 3-16 MBC Read From Memory

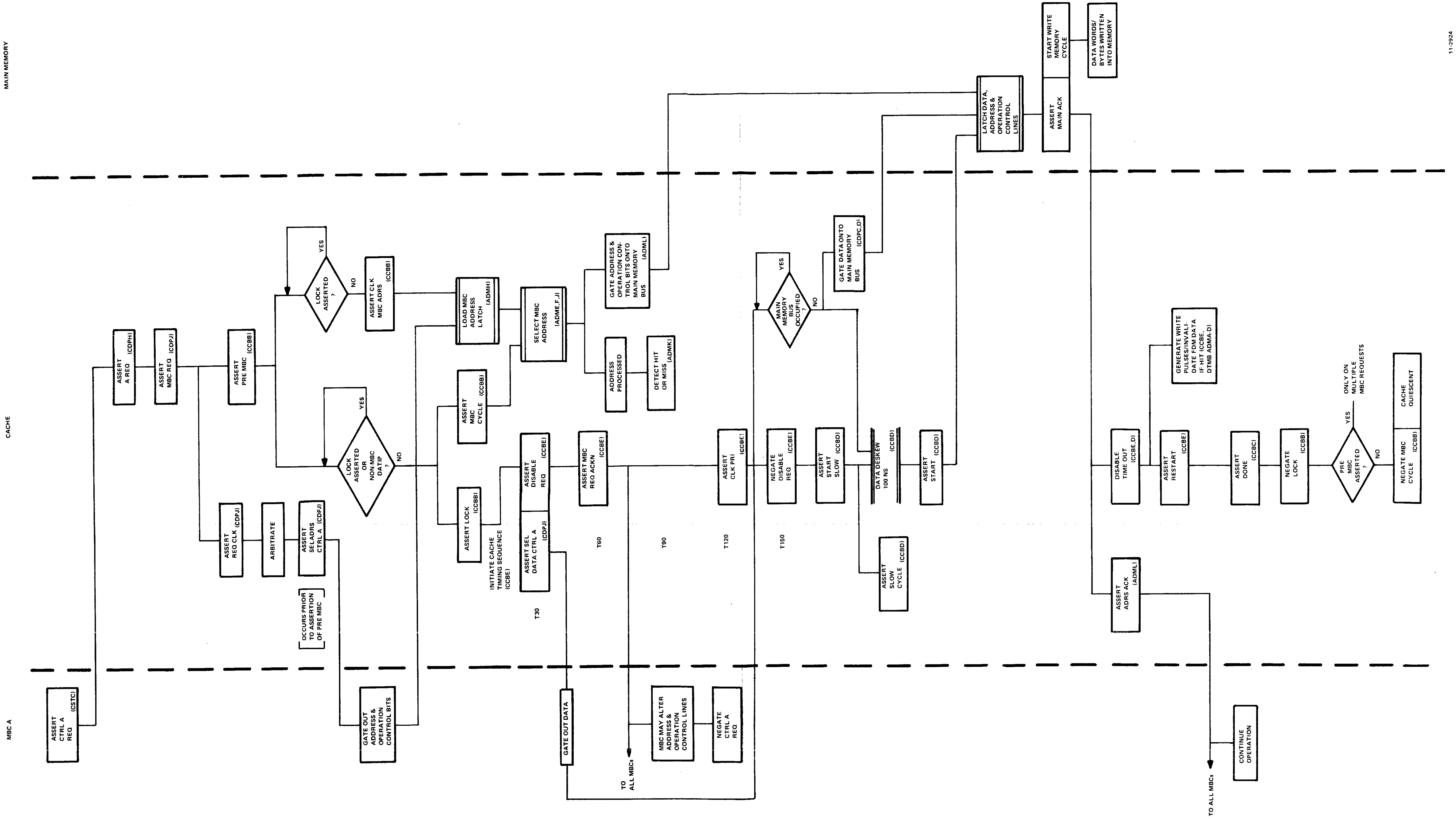

MAIN MEMORY
CACHE
MBC A
11-2924
ASSERT CTRL A REQ (CSTC)
ASSERT A REQ (CDPH)
ASSERT MBC REQ (CDPJ)
ASSERT PRE MBC (CCBB)
ASSERT REQ CLK (CDPJ)
ARBITRATE
ASSERT SEL ADRS CTRL A (CDPJ)
OCCURS PRIOR TO ASSERTION OF PRE MBC
GATE OUT ADDRESS & OPERATION CONTROL BITS
LOCK ASSERTED ?
YES
NO
ASSERT CLK MBC ADRS (CCBB)
LOCK ASSERTED OR NON-MBC DATIP ?
YES
NO
LOAD MBC ADDRESS LATCH (ADMH)
ASSERT MBC CYCLE (CCBB)
ASSERT LOCK (CCBB)
INITIATE CACHE TIMING SEQUENCE (CCBE)
SELECT MBC ADDRESS (ADME,F,J)
GATE ADDRESS & OPERATION CONTROL BITS ONTO MAIN MEMORY BUS (ADML)
ADDRESS PROCESSED
DETECT HIT OR MISS (ADMK)
ASSERT DISABLE REQ (CCBE)
ASSERT SEL DATA CTRL A (CDPJ)
T30
GATE OUT DATA
ASSERT MBC REQ ACKN (CCBE)
T60
T90
TO ALL MBCs
MBC MAY ALTER ADDRESS & OPERATION CONTROL LINES
NEGATE CTRL A REQ
ASSERT CLK PRI (CCBE)
T120
MAIN MEMORY BUS OCCUPIED ?
YES
NO
GATE DATA ONTO MAIN MEMORY BUS (CDPC,D)
NEGATE DISABLE REQ (CCBE)
T150
ASSERT START SLOW (CCBD)
DATA DESKEW 100 NS (CCBD)
ASSERT SLOW CYCLE (CCBD)
ASSERT START (CCBD)
LATCH DATA, ADDRESS & OPERATION CONTROL LINES
ASSERT MAIN ACK
START WRITE MEMORY CYCLE
DATA WORDS/ BYTES WRITTEN INTO MEMORY
DISABLE TIME OUT (CCBE,D)
GENERATE WRITE PULSES/INVALIDATE FDM DATA IF HIT (CCBE, DTMB ADMA-D)
ASSERT RESTART (CCBE)
ASSERT ADRS ACK (ADML)
TO ALL MBCs
CONTINUE OPERATION
ASSERT DONE (CCBC)
NEGATE LOCK (CCBB)
PRE MBC ASSERTED ?
YES
NO
ONLY ON MULTIPLE MBC REQUESTS
CACHE QUIESCENT
NEGATE MBC CYCLE (CCBB)

Figure 3-17 MBC Write to Memory

3.8.10 MBC Write to Memory

The flowchart in Figure 3-17 shows a single MBC (MBC A) requesting memory access. If two or more MBCs request memory access concurrently, Cache operation is similar. Multiple MBC requests are discussed in Paragraph 4.6, where Massbus arbitration is described.

MBCs requesting memory access assert their respective request signals [CSTC CTRL A (B, C, or D) REQ L]. The Cache MBC arbitrator receives these requests and arbitrates among the requesting MBCs. The MBC arbitration logic asserts MBC REQ and transmits SEL ADRS CTRL "X" H to the selected MBC (MBC A in this case). Receipt of the SEL ADRS signal enables MBC A to gate out an address and operation control lines C1, C0, and CX to the Cache. MBC REQ, delayed and synchronized by a Cache internal clock (SYNC CLK), generates PRE MBC (Pre MBC Cycle), which causes the address and control bits gated out by the selected MBC to be loaded into the MBC Address Latch. PRE MBC will initiate Cache operation to service the MBC request *if*:

1. The Cache is not presently servicing the request of some other device (i.e., LOCK not asserted).

2. The Cache is not waiting to execute the write portion of a DATIP initiated by some other device.

3. There are no Unibus Map requests currently pending (i.e., PRE UBUS is not asserted).

If the above conditions are satisfied, the Cache asserts MBC CYCLE and LOCK. LOCK indicates that the Cache is presently "locked" into an operating cycle (MBC CYCLE in this case) and that no other requests will be serviced until the present cycle is completed. MBC CYCLE causes the address generated by the selected MBC to be gated into the Cache by the Address Multiplexer. This address is processed in the Cache, and, at the same time, gated to the Main Memory Bus along with control bits C1 and C0. Incoming address bits (9:2) address the FDM and the Address Memory. Bits (21:10) of the incoming address are checked against the contents of the Address Memory to determine whether the contents of the address referenced are currently stored in the Cache. HIT 0 or HIT 1 will be asserted if the data being requested is in the FDM. If HIT 0 or HIT 1 is asserted, the corresponding data in the FDM will have to be invalidated by loading a negated Valid bit into the Tag 0 Address Memory or Tag 1 Address Memory, respectively.

When LOCK is asserted, a timing sequence is initiated in the Cache. At T30 of the timing sequence, DISABLE REQ is asserted. DISABLE REQ clocks the MBC arbitration logic and causes SEL DATA CTRL A to be transmitted to the selected MBC; this enables the selected MBC to gate write data to the Cache. The write data is gated onto the Main Memory Bus by the Cache when MAIN BOCC becomes unasserted. While DISABLE REQ is asserted, the MBC arbitrator is prevented from arbitrating new incoming requests.

At T60, MBC REQ ACKN is asserted. MBC REQ ACKN is transmitted to all the MBCs and notifies the selected MBC that it can negate its request and alter the address and operation control bits.

At T120 of the Cache timing sequence, CLK PRI H is generated and clocks the MBC priority arbitration logic; this records that MBC A is currently selected and will influence future selections.

At T150 of the Cache timing sequence, DISABLE REQ is negated. DISABLE REQ negated enables clocking the MBC priority arbitration logic, causing selection of the next MBC if an MBC request is pending.

At T180, START SLOW is asserted in the Cache. This asserts SLOW CYCLE and enables the assertion of START on the Main Memory Bus 100 ns after the bus becomes unoccupied (START WRITE asserted). START causes the data, address, and control bits presently on the Main Memory Bus to be loaded into Main Memory. A memory cycle is then started, and MAIN ACK is transmitted back to the Cache. The memory cycle results in the data being written into Main Memory. In response to MAIN ACK, the Cache transmits ADRS ACK to the MBCs; the MBC which initiated the write to memory is thereby notified that the Main Memory operation has been executed.

The receipt of MAIN ACK in the Cache indicates that the Main Memory has responded properly. Therefore, this signal inhibits the generation of a Main Memory Bus time-out.

When time-out is inhibited, a write pulse is generated if a hit has occurred during the cycle. The write pulse loads a negated Valid bit into the Address Memory (Tag 0 or Tag 1), thereby invalidating the FDM data words on which the hit occurred.

When time-out is inhibited, RESTART is asserted, which in turn causes the assertion of DONE. DONE brings the Cache into its quiescent state by negating LOCK, which in turn negates MBC CYCLE. With LOCK negated, the Cache can begin servicing other requests for memory access.

CHAPTER 4
DETAILED LOGIC

4.1 SCOPE

This chapter provides detailed descriptions of Cache logic functions. Paragraph 4.2 provides a detailed description of the Cache data paths. Paragraphs 4.3 through 4.7 describe Cache timing and control logic. Paragraph 4.8 provides Cache register definitions and describes the register logic.

4.2 BLOCK DIAGRAM DESCRIPTION

Figure 3-2 is a block diagram of the Cache, showing the major functional areas of the data paths. Each block on the diagram references the location of the represented logic in the schematics of the engineering print set.

The Cache is implemented on four hex-height modules:

M8142 CCB (Cache Control Board)
M8143 ADM (Address Memory)
M8144 DTM (Data Memory)
M8145 CDP (Cache Data Paths)

The ADM, DTM, and CDP modules contain almost all of the data path logic illustrated in the block diagram. The CCB module contains the register data paths and almost all the timing and control logic in the Cache.

4.2.1 MBC Address Latch

The MBC Address Latch (Drawing ADMH) is clocked by CCBB CLK MBC ADRS L and loaded with an address (MBCBUS A21-A00 L) and operation control bits (MBCBUS C1, C0, CX L) generated by a selected MBC. CCBB CLK MBC ADRS is asserted just prior to the execution of an MBC cycle by the Cache. With the address and operation control bits latched in the Cache, the MBC may unassert or alter these lines upon receipt of CCBE MBC REQ ACKN L from the Cache. The outputs of the MBC Address Latch are routed to the Address Multiplexer.

4.2.2 Address Multiplexer

The Address Multiplexer (Drawings ADME, F, J) multiplexes address and operation control bits from one of four sources to various logic in the Cache. The four sources are:

1. Unibus Map - A 22-bit physical address and C1, C0 operation control lines are selected when a Unibus Map cycle is being executed by the Cache.

2. Processor/Memory Management - A 22-bit. physical address and C1, C0 operation control lines are selected when a processor cycle is being executed by the Cache.

3. MBC Address Latch - A 22-bit address and C1, C0, CX operation control lines, generated by a selected MBC and presently stored in the MBC Address Latch, are selected when an MBC cycle is being executed by the Cache.

4. Power-Up Address Logic - An 8-bit address generated by a counter within the Cache (Drawing ADMJ) is selected during the Cache power-up initialization sequence. During the power-up sequence, this address is incremented from 0 to 255 while, at the same time, it is used to index into the Address Memory. At each address, both Tag 0 Address Memory and Tag 1 Address Memory are loaded with negated Valid bits and correct parity. The negated Valid bits indicate that the address tag fields (and therefore the corresponding data in the FDM) are invalid. This is equivalent to the FDM being empty.

Selection is based on the state of signals CCBB AMS S0 H and CCBB AMX S1 H, which are asserted as follows:

Operation	S1	S0
CP Cycle	0	0
Power-Up	0	1
MBC Cycle	1	0
UB Cycle	1	1

The outputs of the Address Multiplexer are used as follows.

Address Bits (21:10)	Used for comparison with the address fields at the selected Address Memory index position.
Address Bits (09:02)	Used to index into the FDM and Address Memory.
Address Bit 01	Used to select a desired word in the FDM during read or write operations. Also used to select the desired word in Main Memory during write operations.
Address Bit 00	Used to select a desired byte in the FDM and Main Memory during DATOB operations.
C1, C0	Determine the operation to be performed. Used to generate the Main Memory Bus operation control bits MAIN C0:C1 L. Also used to generate the Main Memory Bus Byte Mask bits and address bits A01 and A00.

NOTE

Operation control bits C1, C0 are multiplexed by an extension of the Address Multiplexer, located on Drawing ADMJ.

CX	Used to generate Main Memory Bus Byte Mask bits (during MBC cycles only).
Address Bits (21:02)	The Main Memory Bus operation control bits and Byte Mask bits are gated onto the Main Memory Bus along with a parity bit corresponding to these lines.

4.2.3 Main Memory Bus Control Generator

This circuitry (Drawing ADMJ) generates the Main Memory Bus Byte Mask bits (BYTE MASK 3:0), operation control bit MAIN C0, and the address and control parity bit (ADMJ ADRS PARITY H).

Main Memory Bus operation control bit MAIN C1 is always maintained in the negated state, as illustrated on Drawing ADML. The state of MAIN C0 therefore solely determines whether the Main Memory operation will be a read or a write. MAIN C0 is derived from the C1 operation control bit that is selected as input to the Cache by the Address Multiplexer. If the selected C1 input is negated, ADMJ READ L is asserted, and in turn negates MAIN C0 L on the Main Memory Bus. The MAIN C1:0 signals are thereby encoded for a read operation.

The Byte Mask bits are generated by decoding operation control bits C1:0 (and CX during MBC cycles) and address bits A01:00, which are selected by the Address Multiplexer. The decoding is performed by a pair of type 74S153 dual 4 to 1 multiplexers. If a read operation is to be performed on the Main Memory Bus, ADMJ READ L is asserted. This negates the multiplexer strobe inputs and forces all the multiplexer outputs (ADMJ BYTE MASK 3:0 H) low. During a write operation (ADMJ READ L negated), the multiplexers are strobed and cause Byte Mask bits to be generated as listed in the table on drawing ADMJ.

An address and control parity bit (ADMJ ADRS PARITY H) is generated for the address and control lines of the Main Memory Bus. The parity bit is generated in a slightly unconventional manner, in order to minimize the required logic.

The following signals are input to the final parity generator chip:

1. ADMJ PARA GEN H - This signal is a parity bit for address lines 21:15.

2. ADMJ PARB GEN0 H - This is a Tag 0 Address Memory parity bit for bits 14:10 of the address and the Tag 0 Valid bit (CCBM VALID 0 INPUT L).

3. CCBM VALID 0 INPUT L - This signal counteracts for CCBM VALID 0 INPUT L used in generating ADMJ PARB GEN0 H.

4. ADMJ DATOB H - This signal represents parity for the Byte Mask bits. An odd number (actually one) of Byte Mask bits will be asserted only if this signal is asserted.

5. ADMJ READ H - This signal, used to generate MAIN C0, represents parity for MAIN C1:0, since MAIN C1 is always negated.

6. The remaining input represents parity for bits (09:02) of the Main Memory Bus address.

Main Memory address bits (24:22) are always maintained in the negated state, and therefore are not used in generating the parity bit.

4.2.4 Main Memory Bus Address Drivers

These drivers (Drawing ADML) drive bits (21:2) of the address selected by the Address Multiplexer (along with operation control lines MAIN C1:0) onto the Main Memory Bus. The Cache thus anticipates a cycle to Main Memory whenever the Address Multiplexer makes an address selection. Note that signal MAIN C1 is always maintained in the negated state.

4.2.5 Address Field Inverter

The Address Field Inverters (Drawings ADME, F) perform a simple inversion of bits 21 through 10 to allow for comparison in the parity, address, and validity check circuitry. The inverters compensate for the inversion performed by the Address Memory.

4.2.6 Index Field Inverter-Drivers

The Index Field Inverter-Drivers (Drawings ADMA, C) are used to invert bits (9:2) (index field) of the incoming address, and simultaneously to provide sufficient drive to allow these signals to address all the chips that comprise the Address Memory. The Index Field Inverter-Drivers consist of two sets, as illustrated in the block diagram. One set supplies drive for the Tag 0 Address Memory address inputs (Drawings ADMA, B). The other set performs the same function for the Tag 1 Address Memory (Drawings ADMC, D).

4.2.7 Address Memory

The Address Memory (Drawings ADMA, B, C, D) is comprised of the Tag 0 Address Memory and Tag 1 Address Memory, each containing 256 15-bit address tags. The tags consist of a 12-bit address field, a Valid bit, and two parity bits. The Tag 0 Address Memory (Drawings ADMA, B) contains the address tags for data stored in Group 0 of the FDM, while the Tag 1 Address Memory (Drawings ADMC, D) contains address tags for data stored in Group 1.

The Tag 0 Address Memory consists of 15 type 19-12069 random access memory chips. Each chip stores 1 bit position of the 15-bit address tag. Eight address inputs provide 256 address locations. Data being accessed is available at the Y (pin 6) output. Data to be stored is applied to the DI input (pin 13), and is written by a low pulse at pin 12.

The Tag 1 Address Memory is structured in an identical manner.

The index field of an incoming address (bits 09:02) is used to index into the Tag 0 and Tag 1 Address Memory. The address tags thus accessed are then checked against the address field of the incoming address (bits 21:10) by the Tag 0 and Tag 1 Address, Parity, and Validity Checkers (Drawing ADMK) to determine whether the contents of the incoming address are presently stored in the Cache.

The address memory is written with a new address tag whenever new data is loaded into the FDM as a result of a read miss. Also, whenever an MBC write hit occurs, the Valid bit of the address tag on which the hit occurred is negated; this invalidates the corresponding block in the FDM. (When a non-MBC write hit occurs, the Address Memory is written, but its contents do not change; the address tag written is the same as the old address tag.)

Note that the Tag 0 Address Memory is written whenever data in Group 0 of the FDM is modified or invalidated (ADMJ WP H and CCBM WRITE SEL 0 H is asserted), while the Tag 1 Address Memory is loaded whenever data in Group 1 of the FDM is modified or invalidated (ADMJ WP H and CCBM WRITE SEL 0 H asserted).

4.2.8 Valid Bit Generator

The Valid Bit Generator (Drawing CCBM) generates the Valid bits to be stored in the Address Memory. If data is loaded into the Fast Data Memory as a result of a read miss, the Valid bit associated with the corresponding location in Address Memory is asserted. Two Valid bits are generated by this circuitry: CCBM VALID 0 INPUT L and CCBM VALID 1 INPUT L. They are generated as inputs for the Tag 0 Address Memory and Tag 1 Address Memory, respectively. The circuit used to generate these signals is discussed in detail in Paragraph 4.7.

4.2.9 Address Memory Parity Generator

The Address Memory Parity Generator logic generates the following odd parity bits (Drawings ADMF, J) for loading into the Address Memory:

1. One parity bit (ADMF PARA GEN L) is generated for bits (21:15) of the incoming address, and can be loaded into either the Tag 0 Address Memory or Tag 1 Address Memory.

2. One parity bit (ADMJ PARB GEN 0 H) is generated for bits (14:10) of the incoming address plus the VALID 0 INPUT bit, for loading into the Tag 0 Address Memory.

3. One parity bit (ADMF PARB GEN 1 H) is generated for bits (14:10) of the incoming address plus the VALID 1 INPUT bit, for loading into the Tag 1 Address Memory.

4.2.10 Tag 0 and Tag 1 Parity, Address, and Validity Checker

This logic, represented by two blocks on the block diagram, is located on Drawing ADMK. The Tag 0 Parity, Address, and Validity Checker determines whether valid data, corresponding to the incoming address selected by the Address Multiplexer, is stored in Group 0 of the Fast Data Memory (FDM). The Tag 1 Parity, Address, and Validity Checker performs a parallel check to determine whether valid data corresponding to the same address is stored in Group 1.

The Tag 0 check is performed by comparing address bits (21:10) coming from the Address Multiplexer with address bits (21:10) stored in the Tag 0 Address Memory location selected by incoming address bits (9:2). If bits (21:10) of the incoming and stored addresses match and the Valid bit in the addressed location of the Tag 0 Address Memory is asserted, the Tag 0 Parity, Address, and Validity Checker asserts ADMK GROUP 0 AMATCH H. If no parity errors were detected on the address tag ADMK GROUP 0 PARA OK H and ADMK GROUP 0 PARB OK H are asserted), ADMK GROUP 0 HIT L is asserted. This indicates that the content of the memory address being accessed is presently in Group 0 of the FDM. A hit on Group 0 results in the following:

1. During a non-MBC read, data will be fetched from the FDM at high speed without requiring a slow cycle to Main Memory (unless an FDM parity error is detected on the requested word).

2. During a non-MBC write, the data word in Group 0 of the FDM on which the hit was made will be updated.

3. During an MBC write, the data word in Group 0 of the FDM on which the hit was made will be invalidated by negating the Valid bit stored in the Tag 0 Address Memory location selected by bits (9:2) of the incoming address. Negating this Valid bit invalidates both the odd and even words in Group 0 addressed by bits (9:2) of the incoming address.

Detection of a hit on Group 0 and/or Group 1 can be inhibited by setting bits 4 and/or 5 of the Control Register (17 777 746). This asserts CCBH FORCE MISS GP0 and/or CCBH FORCE MISS GP1, thereby preventing detection of address equality by the comparators.

Setting bits (11:08) of the Maintenance Register (address 17 777 750) forces detection of parity errors by the parity checkers.

4.2.11 Write Data Multiplexer

The Write Data Multiplexer (Drawing CDPE) selects write data from either the BR of the processor or from the Unibus Map, depending on whether a memory access is being performed from the processor or the Unibus Map.

The output of the Write Data Multiplexer is fed to the Main Memory Drivers to be driven across the Main Memory Bus to Main Memory. The data is placed on the Main Memory Bus as soon as the bus becomes vacant (BOCC not asserted and MBC read cycle not being performed).

The write data output of the Write Multiplexer is also applied to the A inputs of the Even Multiplexer and the Odd Multiplexer. During a write operation, the A input is selected by the Even Multiplexer and the Odd Multiplexer, and switched to the Fast Data Memory. The write data thus becomes available to update the data memory if a hit occurs. The output of the Write Data Multiplexer is also applied to the Data Parity Generator, which generates data parity for the word being written, for use on the Main Memory Bus and possible storage in the Fast Data Memory.

During an MBC cycle (CCBB MBC CYCLE L asserted) and during the power-up sequence (CDPJ INIT A L asserted), the select inputs to the Write Data Multiplexer are both high. This forces all 1s to be output from the multiplexer and ensures that the data lines are stable while data parity bits are generated. The all 1s pattern thus generated is written into the FDM during the power-up sequence, and also when an FDM location is invalidated as a result of an MBC write hit.

4.2.12 Data Parity Generator

The Data Parity Generator (lower left of Drawing CDPF) generates odd parity bits (CDPF WRITE MUX LO GEN H and CDPF WRITE MUX HI GEN H) for the two 8-bit bytes gated by the Write Multiplexer. WRITE MUX LO GEN H corresponds to the high byte (WRITE MUX 15:08 H).

The two parity bits are routed to the Main Memory Bus Data Drivers and will be gated onto the Main Memory Bus along with the data from the Write Multiplexer when the write-through operation is performed.

The two parity bits are also routed, along with the data from the Write Multiplexer, to the Even Multiplexer and Odd Multiplexer to allow updating of the Fast Data Memory if a write hit occurs.

4.2.13 Main Memory Bus Data Drivers

These drivers (Drawings CDPC, D) drive the data selected by the Write Data Multiplexer (i.e., processor or Unibus Map data) and the corresponding byte parity bits (generated by the Data Parity Generator from the selected data) onto the data lines of the Main Memory Bus. The 18 bits of data and data parity are driven concurrently on the MAIN DATA BYTE (0-0:0-8) (1-0:1-8) and MAIN DATA BYTE (2-0:2-8) (3-0:3-8) data lines of the Main Memory Bus. Since the processor and Unibus Map are capable of only single word or byte transfers, this arrangement allows writing into either word/byte within a Main Memory block.

Data is gated onto the Main Memory Bus when CDPC CACHE DATA EN H is asserted. This occurs when the Main Memory Bus data lines become unoccupied (CDPD OCC L negated), while a non-MBC write to memory operation is being executed by the Cache.

4.2.14 Main Memory MBC Data Drivers

These drivers (Drawings CDPC, D) drive data (MBCBUS D31–D00 L) and associated byte parity bits (MBCBUS B3PA–B0PA L) from the Massbus Controllers onto the data lines of the Main Memory Bus during a write Cache MBC cycle, when MAIN BOCC becomes unasserted (i.e., when the Main Memory Bus data lines become unoccupied).

Note that the 36 MBCBUS data and data parity lines allow the MBCs to perform double word transfers to and from Main Memory.

4.2.15 Main Memory Bus Data Receivers

The Main Memory Bus Data Receivers (Drawings CDPC, D) are represented by two blocks in the block diagram. One group of receivers receives the low (even addressed) word (i.e., bytes 0 and 1) and the associated byte parity bits that are asserted on the Main Memory Bus. The other group of receivers receives the high (odd addressed) word (i.e., bytes 2 and 3) and the associated byte parity bits that are asserted on the Main Memory Bus. The outputs of the receivers, only used when data is being read directly from Main Memory, are routed directly to the MBCs (for MBC reads) and to the Bus Data (Low Word and High Word) Registers (for non-MBC reads).

4.2.16 Bus Data Register

The Bus Data Register (Drawing CDPA) is active during a non-MBC read miss cycle. The Bus Data Register is loaded with the 36-bit double word being read from the Main Memory when the Cache receives DATA READY asserted on the Main Memory Bus. The Bus Data Register is clocked by CDPA BD CLK H. This signal is asserted in response to DATA READY (CDPC DATA RDY H asserted) provided that an MBC read is not in progress (CDPK RIP L negated).

Data read from Main Memory is always read in two-word pairs by the Cache. Each double word consists of an even word (ADRS BIT 1 = 0) and the next higher odd word (ADRS BIT 1 = 1). These words are stored in the Bus Data (Low Word) Register and the Bus Data (High Word) Register, respectively.

The output of the Bus Data (Low Word) Register is applied to the B inputs of the Even Multiplexer. The output of the Bus Data (High Word) Register is applied to the B inputs of the Odd Multiplexer.

4.2.17 Even Multiplexer and Odd Multiplexer

This logic, located on Drawings CDPB, F, is represented by two blocks on the block diagram. The Even Multiplexer switches data to the even word Fast Data Memory and to the Cache Data Multiplexer.

During a read operation in which a cycle to Main Memory occurred, the Even Multiplexer selects the even word which was brought from Main Memory and is presently stored in the Bus Data (Low Word) Register. This data word is applied to the inputs of the even word Fast Data Memory and will be stored in one of the memory locations. The data word is also applied to the Cache Data Multiplexer (DTMM), and will be transmitted to the device initiating the read operation if the even word was the one requested by the device.

During a write operation, the Even Multiplexer switches the 16 bits of write data from the Write Data Multiplexer, plus two data parity bits, to the even word sections of the Fast Data Memory for possible use in updating the Fast Data Memory. If the write operation is to an address presently stored in the Cache (write hit), the data in the corresponding Fast Data Memory location must be updated if it is to remain valid. If the write operation is to an address not stored in the Cache (write miss), the Fast Data Memory is not updated, and the output of the Even Multiplexer is not used.

The Odd Multiplexer operates in a manner similar to the Even Multiplexer, switching data from the Bus Data (High Word) Register to the Fast Data Memory and Cache Data Multiplexer during a read operation, and switching write data to the odd word sections of the Fast Data Memory during a write.operation.

An extension of the Even and Odd Multiplexers, located at the lower right of Drawing CDPF, switches the byte parity bits.

During the power-up sequence, CDPJ INIT A L is asserted, and causes the Even and Odd Multiplexers to select the all 1s pattern generated by the Write Data Multiplexer.

4.2.18 Main Memory Data Parity Check

This circuitry (Drawing CDPF), represented by two blocks in the block diagram, checks for correct parity on data words brought from Main Memory. The checks are made at the outputs of the Even Multiplexer and Odd Multiplexer, i.e., on the low word and the high word brought from Main Memory. If a parity error is detected, CDPF MAIN LO PAR OK L or CDPF MAIN HI PAR OK L are negated, and the corresponding bits in the Memory System Error Register are caused to set.

Setting bits (15:12) of the Maintenance Register (address 17 777 750), causes byte parity bits to be checked as 1s. This will cause parity errors to be detected on bytes having negated parity bits.

4.2.19 FDM Index Field Drivers

The FDM Index Field Drivers (Drawing DTMA) provide the drive necessary to allow bits (9:2) (index field) of the incoming address to address all the chips that comprise the FDM. The drivers have four sets of outputs (DTMA WRD 0 A09–02 H, DTMA WRD 1 A09–02 H, DTMA WRD 2

A09–02 H, and DTMA WRD 3 A09–02 H). Each set of outputs addresses the chips in the FDM that store a particular word, as listed below.

WRD 0 A09–02 address	The even addressed words in Group 0.
WRD 1 A09–02 address	The odd addressed words in Group 0.
WRD 2 A09–02 address	The even addressed words in Group 1.
WRD 3 A09–02 address	The odd addressed words in Group 1.

4.2.20 Fast Data Memory (FDM)

The 1024 data words that the Cache is capable of storing are stored in the FDM (Figure 2-2). The FDM is divided into two sections or groups (Group 0 and Group 1), each capable of storing 512 18-bit words. The words comprise an eight-bit low byte, a low byte parity bit, an eight-bit high byte, and a high byte parity bit.

Each group is divided into two equal areas (256 words each). In one of the areas, the contents of even addresses (address bit 1 = 0) are stored; the contents of odd addresses (address bit 1 = 1) are stored in the other.

The FDM is implemented using type 19-12069 256 × 1 bit random access memory chips. Nine chips (as shown on Drawing DTMC) can therefore store 256 nine-bit bytes (eight data bits plus one parity bit). The organization on Drawing DTMC is duplicated on Drawings DTMD through DTML. Drawings DTMC–F illustrate Group 0 of the FDM; Drawing DTMH-L illustrate Group 1.

FDM chip select signals DTMB CS3:0L (Drawing DTMB) enable the FDM chips to be written and to output data. The FDM is enabled on a word-by-word basis. DTMB CS0 and CS1 enable the even and odd words (respectively) in Group 0 of the FDM. Similarly, CS2 and CS3 enable the even and odd words (respectively) of Group 1.

During the power-up sequence [ADMJ POWER UP (1) H asserted], all four chip selects are asserted.

During a write cycle (DTMB WRITE H asserted), either DTMB CS0 L and DTMB CS2 L or DTMB CS1 L and DTMB CS3 L are asserted, depending on whether the address being referenced is odd or even (as determined by signal ADME AMX 01 H). Thus, the even word in each FDM group or the odd word in each FDM group will be enabled. If a hit is detected, write pulses will be generated only for the byte/word in the group on which the hit occurred.

During a read hit cycle (CCBD SLOW CYCLE L negated), chip selection is performed in the same manner as during a write cycle.

During a read miss cycle, (ADMJ READ L and CCBD SLOW CYCLE H asserted), either DTMB CS0 L and DTMB CS1 L or DTMB CS2 L and DTMB CS3 L are asserted, as determined by signals CCBM WRITE SEL 0 H and CCBM WRITE SEL 1 H. The CCBM WRITE SEL 1:0 signals are generated by the Group Selection circuitry (Paragraph 4.7).

DTMB LO BYTE WP 0*1 L, DTMB HI BYTE WP 0*1 L, DTMB LO BYTE WP 2*3 L, and DTMB HI BYTE WP 2*3 L are the FDM write pulses. The first pair of signals writes the low and high bytes within Group 0 of the FDM. The second pair writes the low and high bytes in Group 1. FDM chips enabled by a chip select signal are written when the write pulse is low.

NOTE

The chip select signals and the write pulses operate together to write the desired byte, word, or double word into the FDM. the chip select signal must be asserted and the corresponding write pulse must be generated.

During the power-up sequence, CCBM WRITE SEL 1:0 H are asserted; this enables all four write pulses to be generated.

During a read miss, only one of the WRITE SEL signals is asserted; therefore, only the pulses which write into the selected group will be generated.

During a write hit, write pulses are generated for the word/byte in the group on which the hit occurred, as determined by the WRITE SEL signals, ADME AMX00 H and ADMJ DATOB H.

During a write miss, the WRITE SEL signals are negated, and no write pulses are generated.

The contents of a Main Memory location (i.e., a two-word block) is loaded into the FDM whenever a non-MBC read miss occurs. The block is loaded into either Group 0 or Group 1 (depending on the state of an internally generated Random bit; refer to Paragraph 4.7) at an index position determined by the index field (bits 09:02) of the incoming address. The words within the block become available for future reference by either the processor or Unibus Map. When one of these words is read in the near future (assuming that they are not overwritten by another pair of words having an address with an identical index field), it will be fetched at high speed because a Main Memory Bus cycle will not be required.

During a non-MBC write hit, the data word (or byte) written into Main Memory is also written into the FDM. If the word to be written into the FDM has an even address, it is applied via the Even Multiplexer to the even word storage areas of both Group 0 and Group 1 of the FDM. It thus becomes available to replace the obsolete even address word in the group on which the hit occurs.

MBC read hit, read miss, and write miss operations do not affect the FDM. However, when an MBC write hit occurs, the FDM location on which the hit occurred is loaded with all 1s (i.e., correct data parity). At the same time, the corresponding Address Memory location is loaded with a negated Valid bit, invalidating the block.

Within each group of the FDM, the odd word and even word outputs are common collectored. When data is read from the FDM, only the even or odd word in each group is read out. The two words (both odd or both even) are checked for correct parity, and also applied to the Cache Data Multiplexer. During a non-MBC read hit, the Cache Data Multiplexer selects the word from the FDM group (Group 0 or Group 1) on which the hit occurred.

4.2.21 FDM Data Parity Check

This logic (Drawing DTMN), represented by two blocks in the block diagram, checks for correct parity on the data output from Group 0 and Group 1 of the FDM. One parity check is performed on data output from Group 0; the other check is performed on the data output from Group 1. If a parity error is detected on data output from Group 0 or Group 1, DTMN DATA PAR0 OK L or DTMN DATA PAR1 OK L are negated, respectively, and the corresponding bits in the Error Register are set.

If, during a read hit operation, the requested word stored in the FDM is found to have bad parity, the Cache initiates a cycle to Main Memory to fetch the (hopefully error-free) backup copy of the word. The newly fetched word will be loaded into the FDM, replacing the word on which the error occurred.

Setting bits (7:4) of the Maintenance Register (address 17 777 750) causes the FDM data byte parity bits to be checked as 0s. Thus, an FDM data parity error will be detected on bytes having asserted parity bits.

4.2.22 Even and Odd Multiplex Inverters

These inverters (Drawing DTMP) invert the data and data parity bits being read from Main Memory during a non-MBC read miss cycle. The inversion is performed so that all inputs to the Cache Data Multiplexer are at a true high when asserted. The inversion is required to compensate for the extra inversion performed on data being read directly from the Fast Data Memory.

4.2.23 Cache Data Multiplexer

The Cache Data Multiplexer (Drawing DTMM) switches data being read out of the Cache to the BR of the processor and to the Unibus Map. The data switched may be from one of four sources, depending on the memory address requested (odd or even address), and whether a hit occurred on FDM Group 0, Group 1, or neither group.

1. Input A to the Cache Data Multiplexer is the output of Group 1 of the FDM. An odd or even word is gated out of Group 1 of the FDM, depending on whether the address input to the Cache is odd or even. If a hit on Group 1 occurs, this input is selected by the Cache Data Multiplexer.

2. Input B to the Cache Data Multiplexer is the output of Group 0 of the FDM. An odd or even word is gated out of Group 0 of the FDM, depending on whether the address input to the Cache is odd or even. If a hit on Group 0 occurs, this input is selected by the Cache Data Multiplexer.

3. Input C is the even word received by the Cache from Main Memory. If the address input to the Cache is an even address (ADRS BIT 1 = 0) and a read miss occurs, this input is selected by the Cache Data Multiplexer.

4. Input D is the odd word received by the Cache from Main Memory. If the address input to the Cache is an odd ad dress (ADRS BIT 1 = 1) and a read miss occurs, this input is selected by the Cache Data Multiplexer.

The output of the Cache Data Multiplexer becomes available to both the processor and the Unibus Map. If the processor initiated the read operation, the Cache will respond with MEM SYNC to the processor when the data is ready; this will cause the transfer of the data to the BR of the processor.

If the Unibus Map initiated the read operation, the Cache will respond with CCBC UB DONE (Unibus Done) when the data is ready. This will cause the transfer of the data to the data latch in the Unibus Map.

4.2.24 Register Logic

The register logic shown on the Cache data paths block diagram is described in Paragraph 4.8

4.3 CACHE TIMING

The PDP-11/70 Cache operates synchronously with the processor. This synchronous operation aids in achieving overall high operating speeds.

The Cache is synchronized to clock signals generated on the TIG module (M8139). The processor free clock, TIGC TF H, is buffered to generate CCBA ARB CLK H. The buffering circuitry, comprising discrete components (located on Drawing CCBA) is designed for minimal propagation delays and rise and fall times. This is achieved by operating the transistors at or near their active region. CCBA ARB CLK H is inverted by a 74S140 gate to generate CCBA SYNC CLK H. Figure 4-1 illustrates these two waveforms, and emphasizes the 6–10 ns delay introduced by the inversion. Only the negative-going edge of CCBA ARB CLK H is used for clocking purposes. Likewise, only the positive-going edge of CCBA SYNC CLK H is used. Thus the active edges of these two clocks are separated by the 6–10 ns inversion delay.

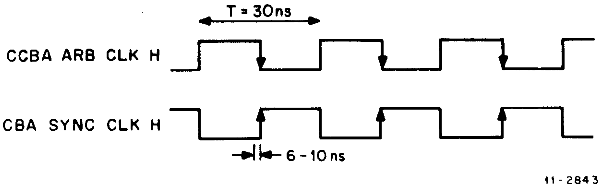


Figure 4-1 Cache Clock Waveforms

The operating speed of the PDP-11/70 Cache is achieved by using fast logic and running the Cache synchronously with the processor. The short time between clock pulses makes Cache timing very critical. To speed up signal processing, a parallel implementation is generally used in place of a serial implementation. For this reason, type 74S64 2-2-3-4 AND-OR-INVERT gates are often used in the design wherever signal delays must be minimized.

4.3.1 Cache Timing Sequence

The cache timing sequence is generated using CCBA ARB CLK H. (Refer to Drawing CCBE.)

The Cache timing sequence is a series of time states generated whenever the Cache begins executing a memory access cycle. These time states are used to synchronize various Cache functions as indicated in Paragraphs 3.8.1 through 3.8.10.

The timing sequence is initiated (Figure 4-2) as a result of the assertion of LOCK. Generation of more than one timing sequence during any Cache cycle is inhibited by gating LOCK with signals CCBD START SLOW (0) H, CCBE T60 (0) H, CCBE T120 (0) H, and CCBE T150 HOLD (0) H.

When LOCK is asserted, the T30 flip-flop is clocked set by the falling edge of ARB CLK. The T60 flip-flop is then clocked set on the next falling edge of ARB CLK. With T60 (1) H asserted, the T30 flip-flop is cleared on the next negative-going ARB CLK pulse. This in turn causes CCBE T90 H to be asserted. On the next negative-going ARB CLK pulse, the T120 flip-flop is set. At the same time, the T60 flip-flop is cleared and CCBE T60 (1) H and CBE T90 H are negated. The T150 HOLD flip-flop is clocked set at the next ARB CLK pulse, while at the same time the T120 flip-flop is cleared. The T150 HOLD flip-flop remains set until either CCBD START SLOW (1) L or CCBC DONE (1) L is asserted. During T150 HOLD, the Cache decides whether or not to assert CCBD START

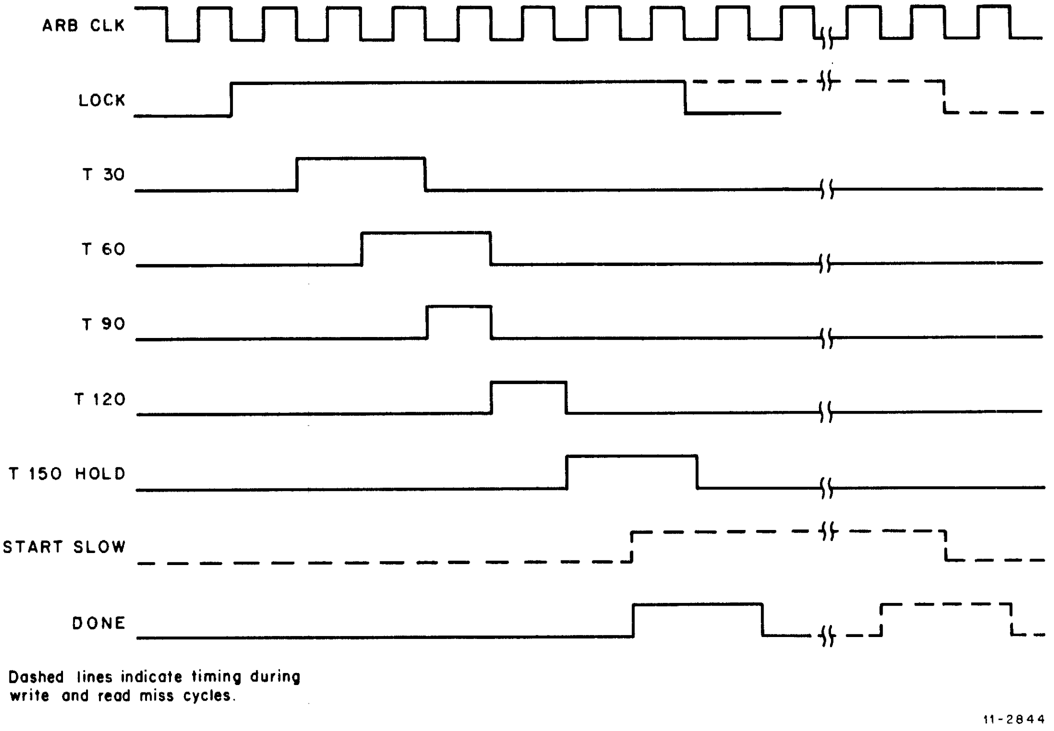

ARB CLK
LOCK
T 30
T 60
T 90
T 120
T 150 HOLD
START SLOW
DONE
Dashed lines indicate timing during write and read miss cycles.
11-2844

Figure 4-2 Cache Timing Sequence

SLOW (1) H and thereby initiate a cycle to Main Memory. Thus if a processor cycle is being performed, the Cache must receive CONTROL OK from the processor during or prior to the assertion of T150 HOLD.

4.3.2 Read Hit Timing

If a read hit is detected during the Cache timing sequence, the Done flip-flop is clocked set (by the first CCBA ARB CLK H pulse during T150 HOLD) and asserts CCBC DONE (1) H. CCBC DONE (1) H resets much of the Cache logic, including the Lock flip-flop. With CCBB LOCK (1) H negated, the Cache is in its quiescent state and may begin executing the next cycle.

If a processor cycle is being executed, CCBC MEM SYNC H is asserted synchronously with processor timing. CCBC MEM SYNC H is asserted at the start of T150 HOLD if the Cache has already received TMCE CONTROL OK H, or during T150 HOLD when the Cache receives TMCE CONTROL OK H.

4.3.3 Main Memory Bus (Slow Cycle) Timing

When the Cache begins executing a processor, Unibus Map, or MBC cycle, bits (21:02) of the incoming address are placed on the Main Memory bus, along with Main Memory control bits (MAIN BYTE MASK 3:0 and MAIN C1:0) and an address parity bit. While the Cache timing sequence is progressing, these signals are deskewed on the Main Memory Bus. During T150 HOLD of the timing sequence, the Cache decides whether or not to perform a Main Memory Bus cycle. If a cycle to Main Memory is required, the Cache asserts CCBD START SLOW. If a write operation is being performed, CDPC START WRITE is asserted when the Main Memory data lines become unoccupied (MAIN BOCC negated). At the same time, write data is gated onto the Main Memory Bus by the Cache. START WRITE asserted enables START SLOW to generate CCBD START H after a 100 ns data deskew delay. CCBD START H is driven onto the Main Memory Bus as MAIN START, and initiates the Main Memory cycle. In response to MAIN ACK from Main Memory, ADML ADRS ACKN H is asserted in the Cache and negates CCBD START H.

During a read operation, CDPC START WRITE is asserted throughout the Cache cycle. This allows CCBD START H to be asserted 100 ns after the assertion of START SLOW.

NOTE

During Main Memory read cycles, the 100 ns delay is still necessary in order to ensure sufficient deskew for the address and control lines of the Main Memory Bus. This is required because the high order bits of the 22-bit physical address generated by Memory Management may not be valid until the Cache is in the midst of its operation cycle.

4.3.4 Timing Restart After Main Memory Cycle

Cache timing is restarted when proper Main Memory response causes the negation of CCBE ALLOW TIMEOUT L. CCBE ALLOW TIMEOUT L is negated when the Cache receives MAIN ACK from Main Memory after initiating a write operation or an MBC read operation. If a non-MBC read operation is being performed, the Cache must also receive MAIN DATA READY. The Cache write pulses (CCBE WP L), which write the Address are generated upon negation of CCBE ALLOW TIMEOUT L.

The negation of CCBE ALLOW TIMEOUT L is synchronized by CCBA SYNC CLK H, CCBA ARB CLK H, and TIGC TF H to generate CCBE MEM SYNC SLOW (1) H and CCBE RESTART (1) H. CCBE MEM SYNC SLOW (1) H causes the assertion of CCBC MEM SYNC H during processor cycles which result in Main Memory operations. CCBE RESTART (1) H enables the assertion of CCBC DONE (1) H. CCBC DONE (1) H resets much of the Cache logic, including the Lock flip-flop. With CCBB LOCK (1) H negated, the Cache is in its quiescent state, and may begin executing the next cycle.

4.4 POWER-UP LOGIC

On power-up, the Cache performs a power-up sequence during which all of the Valid bits in the Address Memory are cleared. This is done because anything stored in the Cache immediately after a power-up must not be construed as valid data. At the same time that the Address Memory Valid bits are negated, all the remaining bits in the Address Memory and FDM are loaded with bit patterns having correct parity. This is to ensure that the bit patterns resident in the Address Memory and FDM upon power-up will not generate parity errors when program execution begins.

The power-up control logic is located on Drawing ADMJ. The circuitry consists of a pulse generator, a counter, and the PUP (Power-Up) flip-flop.

The following discussion describes operation of the power-up circuitry upon initial power turn on. It should, however, be kept in mind that the same sequence of events occurs when power returns after a momentary failure.

Figure 4-3 is a timing diagram illustrating the power-up circuitry operation.

As ac power is appearing at the power supply inputs, AC LO L is asserted and clears the eight-bit Power-Up Address Counter. When DC LO L is asserted, the Power-Up flip-flop is direct set; this enables the Power-Up Pulse generating oscillator to begin operation when power has reached normal levels (i.e., when AC LO L is negated).

When POWER UP (1) H is asserted, the following events occur:

1. CCBB AMX S0 H is asserted, causing the Address Multiplexer to select the power-up address generated by the Power-Up Address Counter. (During power-up, CCBB AMX S1H is unasserted due to INIT.)

2. CCBM VALID 0 INPUT L and CCBM VALID 1 INPUT L are negated, and CCBM WR OK H, CCBM WRITE SEL 0 H, and CCBM WRITE SEL 1 H are asserted.

3. DTMB GROUP 0 H and DTMB GROUP 1 H are asserted and, in turn, enable assertion of DTMB LO BYTE WP0*1 L, DTMB HI BYTE WP0*1 L, DTMB LO BYTE WP2*3 L, and DTMB HI BYTE WP2*3 L when write pulses are generated.

4. The FDM chip selects (DTMA CS 0 L, DTMA CS 1 L, DTMA CS 2 L, and DTMA CS 3 L) are asserted.

5. The output of the Write Multiplexer (CDPE) is forced to all ones.

6. The Even Multiplexer and Odd Multiplexer (CDPB,F) select the outputs of the Write Multiplexer.

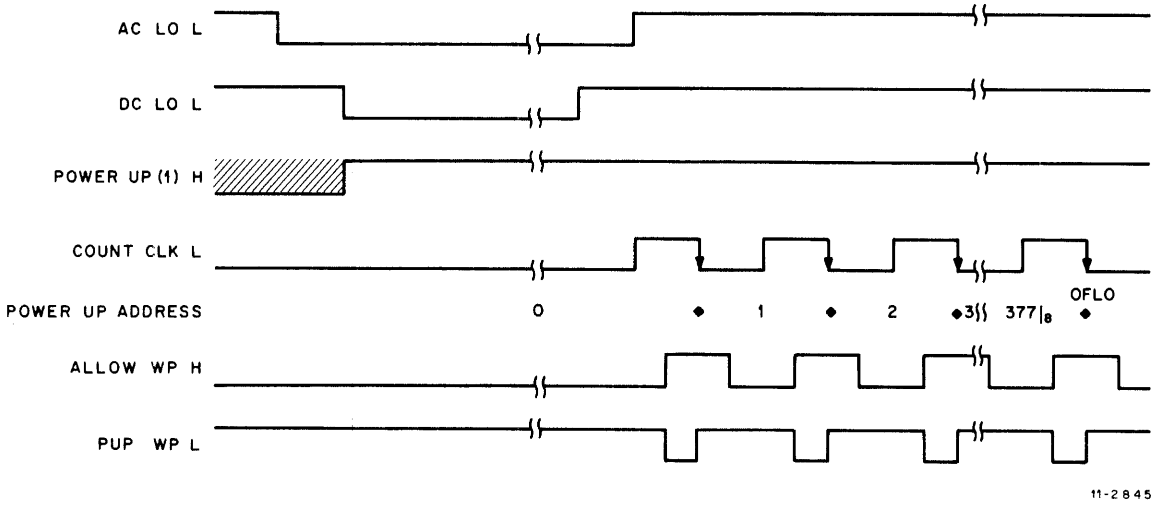


Figure 4-3 Power-Up Sequence Timing Diagram

When AC LO L is negated, the oscillator begins generating pulses. The pulses produced clock the FDM and Address Memory (causing the locations indexed by the Power-Up Address Counter to be loaded) and increment the Power-Up Address Counter. Each FDM word position indexed by the Power-Up Address Counter is loaded with all ones. The Address Memory word locations are loaded with negated Valid bits and correct address parity. As the Power-Up Address Counter is clocked from 000 to 377_8, all the locations in the FDM and Address Memory are loaded, and the contents of the Cache are thereby invalidated.

When the Power-Up Address Counter is clocked to overflow, the Power-Up flip-flop is clocked clear; this inhibits further PUP WP L pulses and terminates the power-up sequence.

4.5 REQUEST ARBITRATOR LOGIC

The Request Arbitrator (Drawing CCCBB) determines whether the Cache will perform a processor, Unibus Map, or MBC memory access.

Two request signals are input to the Request Arbitrator:

1. MAPF UB REQUEST (1) L from the Unibus Map.
2. CDPJ MBC REQ L from the MBC Arbitration Logic.

The request signals are synchronized and delayed (90 ns and 180 ns, respectively) and then assert PRE UBUS and/or PRE MBC. PRE UBUS causes the UBUS flip-flop to be set when LOCK is negated, provided that no non-Unibus DATIPs are in progress. PRE MBC causes the MBC flip-flop to be set when LOCK is negated, provided that no non-MBC DATIPs are in progress.

If the UBUS flip-flop is set, CCBB UB CYCLE H is asserted; the Cache will perform a Unibus Map memory access cycle. If the MBC flip-flop and the UBUS flip-flop are set, CCBB UB CYCLE H is asserted, and the Cache will perform a Unibus Map memory access cycle; this is what gives Unibus Map requests priority over MBC requests. CCBB MBC CYCLE is asserted when the MBC flip-flop is set and the UBUS flip-flop is not; the Cache would then perform an MBC memory access cycle.

Processor memory access cycles are performed only when neither the Unibus Map nor the MBCs are requesting memory access. When neither the UBUS flip-flop nor the MBC flip-flop is set, CCBB CP CYCLE H is asserted. This is a default condition, and therefore gives the processor the lowest priority status. The priority structure is thus:

1st priority: Unibus Map
2nd priority: MBCs
3rd priority: Processor

Whenever a memory access cycle is performed, the CCBB AMX S1, S0 H signals are generated to enable the Address Multiplexer (ADMH) to select address and operation control bits from the correct source.

4.6 MBC ARBITRATION LOGIC

The MBC Arbitration Logic, located on the CDP module (Drawings CDPH through CDPK), selects one of four possible Massbus Controllers and performs with it the protocol required to transfer data on the RH70-Cache Interface.

As illustrated in Figure 4-4, the MBC Arbitrator can be considered a discrete device which just happens to be located on the Cache modules.

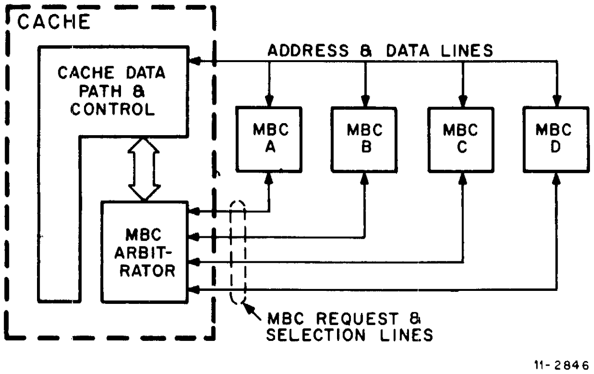


Figure 4-4 Relationship of the MBC Arbitrator to the Cache

The selection of an MBC is based on a number of criteria:

1. If an MBC DATIP/DATO memory cycle is not in proggress and no MBC requests are pending or being executed, thhe first request received will be granted.
2. If an MBC is performing a DATIP/DATO memory cycle, requests from other MBCs are not recognized until the DATO portion of the DATIP/DATO cyycle has been initiated.
3. Jumpers (W1, W2, W3) on the CDP module allow MBC selection to be based on the history of the most recent selections. Table 4-1 lists the selection patterns obtainable for the different jumper configurations. If two or more MBCs request memory access concurrently, selection will be based on the pattern of previous selections, in a manner determined by the jumper configuration.

Table 4-1
MBC Selection Priorities

Jumper Configuration			Priority Structure *
W1	W2	W3	
OUT	OUT	OUT	(A↔B)↔(C↔D)
OUT	OUT	IN	(A→B)↔(C↔D)
OUT	IN	OUT	(A↔B)↔(C→D)
OUT	IN	IN	(A→B)↔(C→D)
IN	OUT	OUT	(A↔B)→(C↔D)
IN	OUT	IN	(A→B)→(C↔D)
IN	IN	OUT	(A↔B)→(C→D)
IN	IN	IN	(A→B)→(C→D)

*SYMBOLS↔, → are defined in the text.

Figure 4-5 is a block diagram showing the three major sections of the MBC Arbitration Logic: Request Block Logic, Address and Data Select Logic, and Data Ready Logic.

4.6.1 Request Block Logic (Drawing CDPH)

Requests from the RH70s are input to the Request Block Logic. If one of the MBCs is currently performing a DATIP, this circuitry inhibits requests of other MBCs from being processed by the Cache. Although none of the Massbus devices presently manufactured by DEC utilize DATIP cycles, the Request Block circuitry has been implemented to cover their utilization in the future.

4.6.2 Address and Data Select Logic

Refer to Drawing CDPJ and Figure 4-6.

4.6.2.1 Single Request Operation – The Address and Data Select Logic input latch (consisting of the four D-type flip-flops at the left-hand side of Drawing CDPJ) is clocked when MBC REQ L is asserted. This causes SEL ADRS CTRL X H (where X is A, B, C, or D) to be transmitted to the requesting MBC. At T30 of the Cache MBC cycle, the Address and Data Select output latch (a type 74S175 data latch at the center of Drawing CDPJ) is clocked by DISABLE REQ H asserted, and SEL DATA X H is transmitted to the selected MBC.

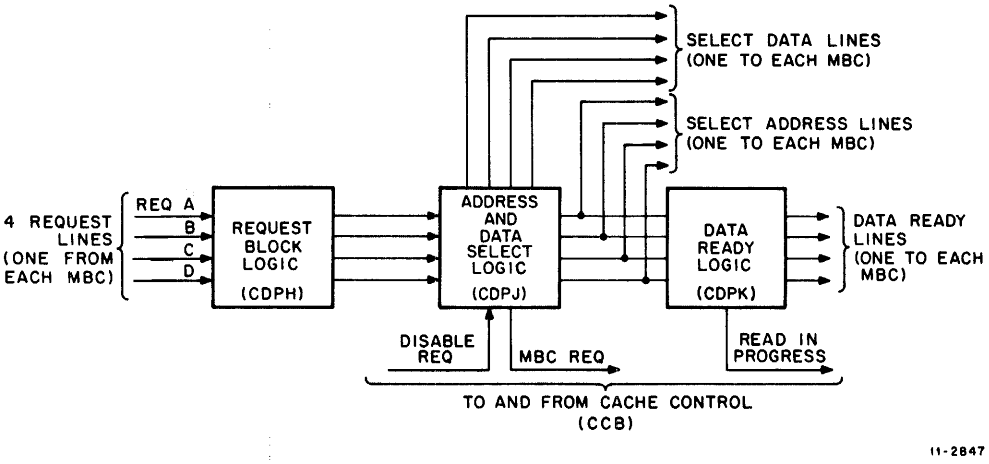
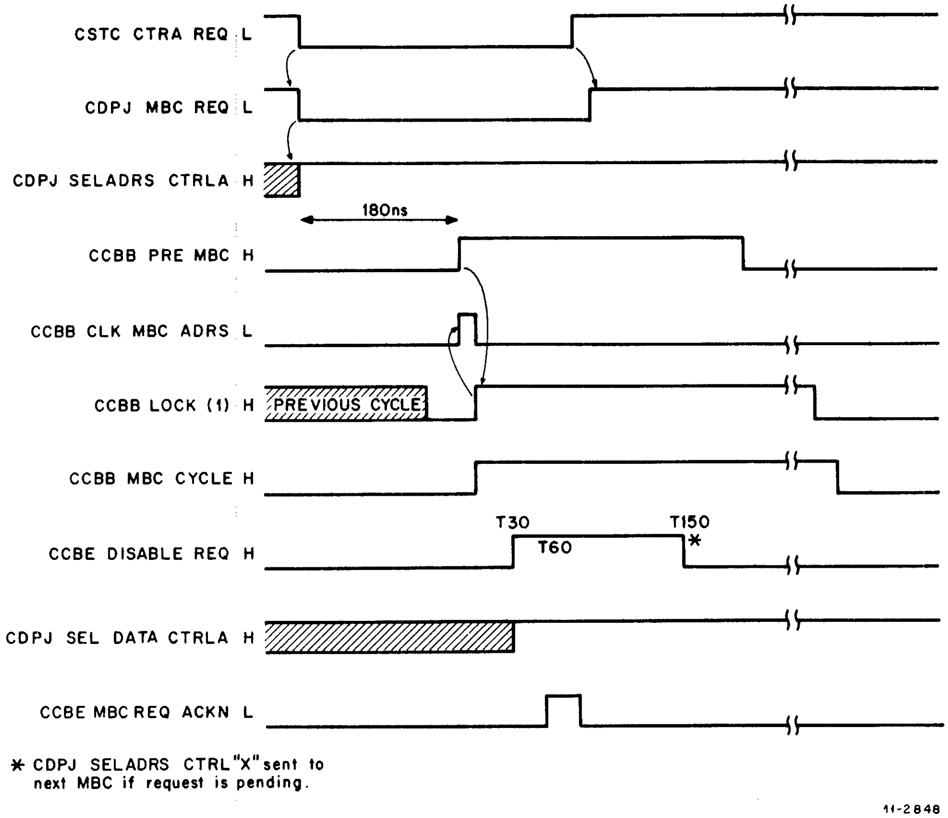


Figure 4-5 MBC Arbitrator Block Diagram



* CDPJ SELADRS CTRL"X"sent to next MBC if request is pending.

Figure 4-6 MBC Request Timing (MBC A Requesting)

When CCBE CLK PRI H is asserted at T120, the Priority Generator (upper-right of Drawing CDPJ) is clocked and records the current MBC selection. Future selections are based on the output of the Priority Generator.

4.6.2.2 Multiple Request Operation - Multiple MBC requests are handled in a manner similar to a single request. In fact, the first of the multiple requests to arrive is always serviced first, and is handled in the same way as a single request. The remaining requests are handled slightly differently because CDPJ MBC REQ L remains asserted; this enables the Address and Data Select Logic input latch to be clocked by the trailing edge of DISABLE REQ H at T150 of the MBC cycle. The input latch is thus loaded with any MBC requests still pending.

Figure 4-7 is a timing diagram of the Address and Data Select Logic during multiple request operation. It is assumed that a straight priority structure (A over B ovver C over D) has been selected via the priority jumpers, and that the MBC requests arrive in the following sequence: B, C, A. The MBC B request, first to arrive, asserts MBC REQ, which in turn asserts SEL ADRS CNTR B. Therefore, the MBC B request is serviced first. The MBC A request is serviced next because of the straight priority struuucture, and finally the MBC C request is serviced.

The Priority Generator determines which request will be granted when more than one MBC request is pending. Jumpers W1, W2, and W3 allow control of the priority structure. When all jumpers are out, a pseudo round-robin priority structure results as follows:

(A↔B) ↔ (C↔D)

The symbol ↔ indicates that selection alternates between the expressions on either side of the symbol.

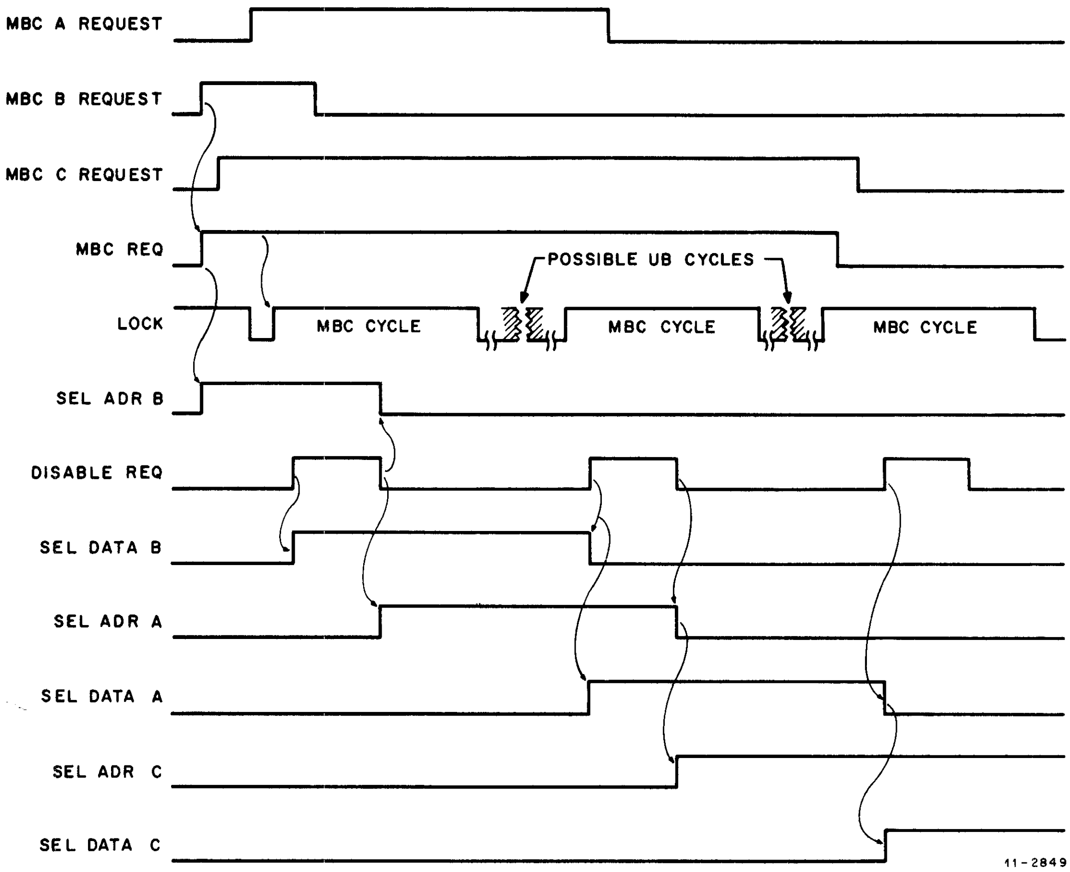


Figure 4-7 MBC Address and Data Select Timing (Multiple Requests – Straight Priority)

With all three jumpers out, the signals output by the Priority Generator indicate the true history of past MBC selections, as follows:

CDPJ B LAST (1) H — MBC A has not been selected since MBC B was last selected.

CDPJ D LAST (1) H — MBC C has not been selected since MBC D was last selected.

CDPJ C OR D LAST (1) H — MBC A or B have not been selected since MBC C or D was last selected.

Each of the above signals can be forced to assertion by installing jumpers W1, W2, or W3, respectively. When jumpers are installed, MBC selection is based on a distorted view of past history, and therefore, some MBCs ccan be given priority over others. For example, if all the jumpers are installed, a straight priority structure results:

(A→B) → (C→D)

where the symbol → indicates that the expression on the left is given priority over the expression on the right.

Table 4-1 lists the MBC selection priorities resulting from the eight jumper configurations.

4.6.3 Data Ready Logic

The Data Ready Logic (located on Drawing CDPK) keeps trackk of which MBCs are currently performing read operations and routes the DATA RDY signal originating in Main Memory to the correct MBC. A read operation can be initiated on the Main Memory Bus before a previous read operation has been completed. This is termed "stacking" operations on the Main Memory Bus. The Data Ready Logic can keep track of two concurrent reads, thereby allowing two MBC read operatiions to be stacked on the Main Memory Bus.

Refer to Drawing CDPK. When an MBC read cycle is performed by the Cache, one of the flip-flops at the left-hand side of the drawing is direct set at T300 of the Cache timing sequence. If no other MBC is currently performing a read from memory (i.e., waiting for data ready), CDPK RIP H (Read In Progress) is in the negated state. This allows one of the flip-flops in the center of the drawing to be direct set, causing the assertion of CDPK RIP H.

As a specific example, assumme that MBC A performs a read from memory. When the Cache asserts CCBE MBC REQ ACKN L at T30 of the Cache timing sequence, the flip-flop at the top left of CDPK is direct set. If no other MBC read is in progress (RIP negated), the top center D-type flip-flop is also set. This enables the top 74S11 to gate DATA RDY originating in Main Memory control to MBC A. Assume, however, that before Main Memory responds with DATA RDY, the Cache begins executing a read from memory initiated by MBC D. When thhe Cache asserts CCBE MBC REQ ACKN at T30 of the current Cache timing sequence, the D-type flip-flop at the lower left of CDPK is direct set. The flip-flop at the lower center is not direct set at this time because CDPPK RIP H is asserted; the Cache is waiting for Main Memory to respond with DATA RDY to a read initiated by MBC A. When DATA RDY is received in the Cache, CDPK DATA RDY CNTL A is asserted and routed to MBC A. At the trailing edge of DATA RDY, the RIPP A flip-flop is clocked clear. This negates RIP H momentarily, and allows the RIP D flip-flop to be direct set. RIP H is thereby reasserted, and the transmission of CDPK DATA RDY CNTL D is enabled.

The Cache thus keeps track of two concurrent MBC reads, and remembers for which MBC a DATA RDY response from Main Memory is intended.

A third MBC read cycle is inhibited by the assertion of CCBD READ IN PROG (1) H. This signal is asserted during the second of two stacked MBC reads and inhibbits negation of CCBE ALLOW TIMEOUT L until the ffirst MBC read is terminated.

4.7 GROUP SELECTION AND VALID BIT LOGIC

The Group Selection and VValid Bit Logic is located on Drawing CCBM. The group selection circuitry produces outputs CCBM WRITE SEL 0 H and CCBM WRITE SEL 1 H. These signals enable Group 0 and Group 1 of the FDM and corresponding Tag 0 or Tag 1 Address Memory to be written.

The Valid Bit Logic asserts CCBBM VALID 0 INPUT L and CCBM VALID 1 INPUT L, the Valid bit inputs to the Tag 0 and Tag 1 Address Memoryy.

The heart of the circuitry is the D-type flip-flop at the lower left of Drawing CCBM. This is the Random bit generator. At T60 of every Cache timing sequence, the Random flip-flop is clocked and causes the Random bit to change state. During normal error-free operation, the state of the Random bit determines which group of the FDM is loaded when a non-MBC read miss occurs. If CCBM RANDOM (1) H is asserted during a non-M'BCC read miss cycle, CCBM WRITE SEL 1 H and CCBM VALID 1 INPUT L are asserted. Likewise, if CCBBM RANDOM (1) H is negated during a non-MBC read miss cycle, CCBM WRITE SEL 0 H and CCBM VALID 0 INPUT L are asserted.

A four-bit latch (type 74175) is clocked by CCBD START SLOW (1) H just prior to the initiation of a slow cycle on the Main Memory Bus. The outputs of this latch represent conditions detected in the FDM and Address Memory during the Cache timing sequence. The R3(1) output will be high if a parity error has been detected in either Group 1 of the FDM or Tag 1 Address Memory. The R2(1) output will be high if a parity error has been detected in either Group 0 of the FDM or Tag 0 Address Memory. The R1(1) output will be low if a write hit on Group 0 has been detected. The R0(1) outpput will be low if a write hit on Group 1 has been detected. If any of the above conditions is detected, the Random bit is overridden. If a parity error is detected, the group in which the error occurred is selected for replacement. On a write hit, the group on which the hit occurs is selected for replacement.

The Random bit is also overridden during the power-up sequence and during MBC cycles. During a power-up, ADMJ POWER UP (1) L asserted causes the assertion of CCBM WRITE SEL 0 H and CCBM WRITE SEL 1 H, and the negation of CCCBM VALID 0 INPUT L and CCBM VALID 1 INPUT L. During an MBC cycle, the Valid bits are also negated, while the assertion of CCBM WRITE SEL 0 H and WRITE SEL 1 H is inhibited if the cycle is a read from memory.

4.8 CACHE REGISTERS AND REGISTER LOGIC

This section defines the Cache registers and the bits they contain; a description of the actual implementation is also provided.

Table 4-2 lists the six registers located in the Cache, along with their addresses. The following paragraphs describe each register.

Table 4-2
Cache Registers

Register	Address	Access
Low Error Address	17 777 740	Read only
High Error Address	17 777 742	Read only
Memory System Error	17 777 744	Read/selective clear
Control	17 777 746	Read/write
Maintenance	17 777 750	Read/write
Hit/Miss	17 777 752	Read only

4.8.1 Low Error Address Register (17 777 740)

This register, illustrated in Figure 4-8, contains the 16 ~~166~~ low order bits of the 22-bit physical address being accessed when an error occurred. The least significant bit is bit 0. The high order bits of the addresses are contained in the High Error Address Register.

All bits are read only. The bits are undetermined after a power-up. They are not affected by a Console Start or RESET instruction.

4.8.2 High Error Address Register (17 777 742)

This register, illustrated in Figure 4-9, ccontains the six high order bits of the 22-bit physical address being accessed when an error occurred. The type of memory cycle being performed when the error occurred is indicated by register bits 15 and 14, which store the operation control bits (C1 and C0) of the memory cycle. Table 4-3 lists the register bits.

All the bits are read only. The bits are undetermined after a power-up. They are unaffected by a Console Start or RESET instruction.

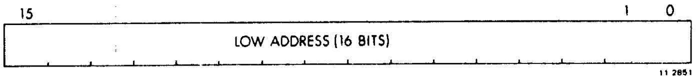


Figure 4-8 Low Error Address Register

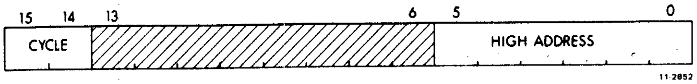


Figure 4-9 High Error Address Register

Table 4-3
High Error Address Register

Bit	Name	Function
15–14	Cycle Type	These bits are used to encode the type of memory cycle which was being requested when the parity error occurred. Bit 15 Bit 14 Cycle Type 0 0 Data In (read) 0 1 Data In Pause 1 0 Data Out 1 1 Data Out Byte
5–0	Address	These bits contain the highest 6 bits of the 22-bit address of the first error. The most significant bit is bit 5.

4.8.3 Memory System Error Register (17 777 744)

The Memory System Error Register, illustrated in Figure 4-10, keeps track of hard and soft errors within the memory system.

A soft error is an error which does not result in the processor receiving erroneous data; a soft error causes a trap.. An error which causes the processor to receive erroneous data is a hard error; this type of error causes an abort.

Table 4-4 defines the bits in the Memory System Error Register. All the bits are read/write. The bits are cleared on power-up or by Console Start: They are unaffected by a RESET instruction.

When writing to the Memory System Error Register, a bit is unchanged if a 0 is written to it, and it is cleared if a 1 is written to it. Thus, the register is cleared by writing the same data back to the register. This guarantees that if additional error bits were set between the read and the write, they will not be inadvertently cleared.

4.8.4 Control Register (17 777 746)

This six-bit register, illustrated in Figure 4-11, controls several important internal functions; these are outlined in Table 4-5. The Control Register allows running thhe PDP-11/70 in a degraded mode; this may be desirable if parts of the Cache are malfunctioning. If Group 0 of the Cache is malfunctioning, it is possible to force all operations through Group 1. Setting bit 4 or bit 5 allows the internally generated Random bit to be overridden and causes data, fetched from Main Memory as a result of a read miss, to be replaced in the specified group. If bits 5 and 2 of the Control Register are set and bits 4 and 3 are cleared, the CPU will not be able to read data from Group 0, and all Main Memory data replacements will occur within Group 1. In this manner, half the Cache will be operating. Bus system throughput will not decrease by 50 percent, since the statistics of read hit probability will still provide reasonably fast operation. If Group 1 is malfunctioning, bits 4 and 3 should be set and bits 5 and 2 cleared so that only Group 0 is operating. If all of the Cache is malfunctioning, bits 3 and 2 should be set. The Cache will be bypassed, and all references will be to Main Memory.

Control Register bits 5 and 4 can also be used to keep a desired routine in the Fast Data Memory. For example, if bit 5 is cleared and bit 4 is set prior to execution of a desired routine, the routine will be loaded into Group 0. If bit 5 is cleared and bit 4 is set when the desired routine is not being executed, the routine will remain protected in Group 0 for future reference. The routine can be protected in Group 0 while it is being executed if bit 5 is set and bit 4 is cleared.

Table 4-6 summarizes the uses of Control Register bits (5:2).

Bits 1 and 0 can be set to disable trapping. With these bits set, the processor will not spend time performing trap service routines each time a non-fatal error occurs. Overall system operation will produce correct results; however, more Main Memory Bus cycles may be performed.

The Control Register can also be used in troubleshooting. For example, by setting register bits 3 and 2, the Cache is effectively disabled. If the system operates with these bits set and does not operate if they are cleared, a malfunction in the Cache is indicated.

Bits (5:0) are read/write. The bits are cleared on power-up or by Console Start. They are unaffected by a RESET instruction.

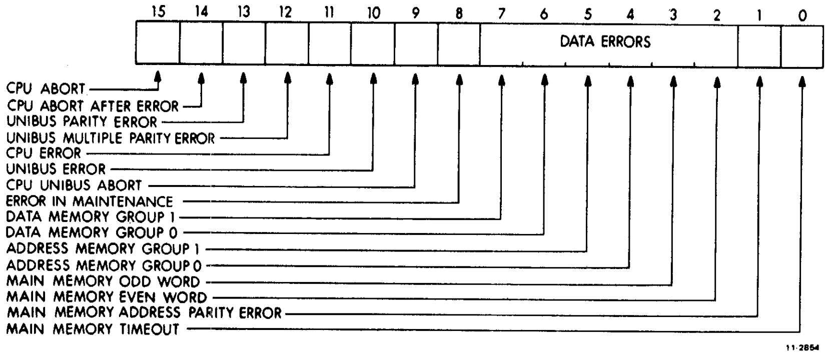


Figure 4-10 Memory System Error Register

Table 4-4
Memory System Error Register

Bit	Name	Function
15	CPU Abort	Set if an error occurs which causes the Cache to abort a processor cycle.
14	CPU Abort After Error	Set if an abort occurs with the Error Address Register locked by a previous error.
13	Unibus Parity Error	Set if an error occurs which results in the Unibus Map asserting the parity error signal on the Unibus.
12	Unibus Multiple Parity Error	Set if an error occurs which causes the parity error signal to be asserted on the Unibus with the Error Address Register locked by a previous error.
11	CPU Error	Set if any memory error occurs during a Cache cycle from the processor.
10	Unibus Error	Set if any memory error occurs during a Cache cycle from the Unibus.
9	CPU Unibus Abort	Set if the processor traps to vector 114 because of a Unibus parity error on a DATI or DATIP cycle by the processor on the Unibus.
8	Error in Maintenance	Set if an error occurs when any bit in the Maintenance Register is set. The Maintenance Register will then be cleared.
7–6	Data Memory	These bits are set if a parity error is detected in the Fast Data Memory in the Cache. Bit 7 is set if there is an error in Group 1, bit 6 for Group 0.
5–4	Address Memory	These bits are set if a parity error is detected in the Address Memory in the Cache. Bit 5 is set if there is an error in Group 1, bit 4 for Group 0.
3–2	Main Memory	These bits are set if a parity error is detected on data from Main Memory. Bit 3 is set if there is an error in either byte of the odd word, bit 2 for the even word. An abort occurs if the error is in the word needed by a CPU reference. A trap occurs if the error is in the other word, or if it is a Unibus reference.
1	Main Address Parity Error	Set if there is a parity error detected on the address and control lines on the Main Memory Bus.
0	Main Memory Time-out	Set if there is no response from Main Memory. For CPU cycles, this error causes an abort. When a Unibus device requests a non-existent location, this bit will not set.

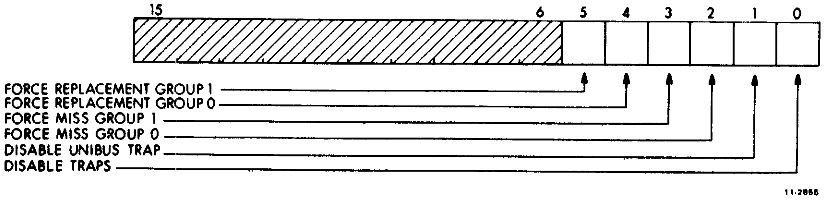


Figure 4-11 Control Register

Table 4-5
Control Register

Bit	Name	Function
5–4	Force Replacement	Setting these bits forces data replacement within a group in the Cache by Main Memory data on a read miss. Bit 5 selects Group 1 for replacement; bit 4 selects Group 0.
3–2	Force Miss	Setting these bits forces misses on reads to the Cache. Bit 3 forces misses on Group 1; bit 2 forces misses on Group 0. Setting both bits forces all cycles to Main Memory.
1	Disable Unibus Trap	Set to disable traps to vector 114 when the parity error signal is placed on the Unibus.
0	Disable Traps	Set to disable traps from soft errors.

4.8.5 Maintenance Register (17 777 750)

This register, illustrated in Figure 4-12, is used for memory system maintenance. Table 4-7 lists the functions of the register bits.

The Maintenance Register is read/write. It is cleared on power-up or by Console Start. It is also cleared whenever any memory system error is detected.

This register is for maintenance use only.

4.8.6 Hit/Miss Register (17 777 752)

The Hit/Miss Register, illustrated in Figure 4-13, indicates whether the six most recent references by the CPU were hits or misses. A one indicates a read hit; a zero indicates a read miss or a write. The lower numbered bits are for the more recent cycles.

All the bits are read only. The bits are undetermined after a power-up. They are not affected by a RESET instruction.

This register is for maintenance use only.

Table 4-6
Control Register Bits 5:2

Control Register Bits		Bit Patterns				
5	Force Replacement to Group 1	1	0	X	1	0
4	Force Replacement to Group 0	0	1	X	0	1
3	Force Miss on Group 1	0	1	1	0	0
2	Force Miss on Group 0	1	0	1	0	0
FUNCTION		Disables Group 0	Disables Group 1	Disables Cache (Group 0 and 1)	Protects and maintains code in Group 0 while it is executed	Protects and maintains code in Group 1 while it is executed

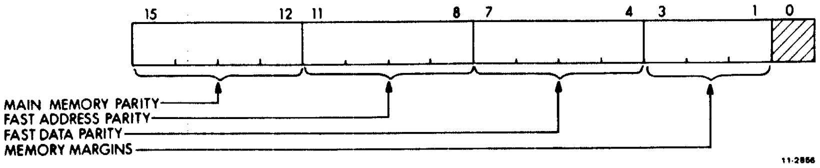


Figure 4-12 Maintenance Register

4.8.7 Use of Cache Registers

When a memory system error is detected, the processor traps to location 114. If location 114 is used as a trap catcher, the operator can examine the Memory System Error Register to determine the type of error which has occurred. The Low Error Address and High Error Address Registers can then be examined to determine where in the program, and during what type of cycle, the error occurred. If statistics on the hit ratio are desired, the Hit/Miss Register can be read. The Control Register can be read to determine what the control conditions were at the time the error occurred.

If location 114 is not used as a trap catcher, the above tasks must be performed by the trap service routine.

If bit 14 (CPU Abort After Error) or bit 12 (Unibus Multiple Parity Error) of the Memory System Error Register is set, the address stored in the Low Error Address and High Error Address Registers is the address of the first error and not the address at which the most recent error occurred. The address at which the most recent error occurred must be reconstructed from the contents of the SP (which points to the virtual address incremented by 2) and the appropriate Memory Management PAR.

The contents of the Memory System Error Register and the High and Low Error Address Registers indicate the failing section of the memory system.

For example, if type MJ11 16K core memory is used in the system, and a Main Memory parity error bit is set in the Error Register, all the information required to determine the failing 16K section

Table 4-7
Maintenance Register

<table>
<tr><th>Bit</th><th>Name</th><th>Function</th></tr>
<tr><td>15–12</td><td>Main Memory Parity</td><td>Setting these bits causes the four Main Memory parity bits to be checked as 1s.
There is one bit per byte; there are four bytes in the data block.
<table>
<tr><th>Bit Set</th><th>Byte</th></tr>
<tr><td>15</td><td>Odd word, high byte</td></tr>
<tr><td>14</td><td>Odd word, low byte</td></tr>
<tr><td>13</td><td>Even word, high byte</td></tr>
<tr><td>12</td><td>Even word, low byte</td></tr>
</table></td></tr>
<tr><td>11–8</td><td>Fast Address Parity</td><td>Setting these bits causes the four parity bits for fast address memory to be wrong.
Bits 11 and 10 affect Group 1; bits 9 and 8 affect Group 0.</td></tr>
<tr><td>7–4</td><td>Fast Data Parity</td><td>Setting these bits causes the four parity bits to be checked as 0s.
<table>
<tr><th>Bit Set</th><th>Byte</th></tr>
<tr><td>7</td><td>Group 1, high byte</td></tr>
<tr><td>6</td><td>Group 1, low byte</td></tr>
<tr><td>5</td><td>Group 0, high byte</td></tr>
<tr><td>4</td><td>Group 0, low byte</td></tr>
</table></td></tr>
<tr><td>3–1</td><td>Memory Margins</td><td>These bits are encoded to do maintenance checks on Main Memory.
<table>
<tr><th>Bit 3</th><th>Bit 2</th><th>Bit 1</th><th></th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Normal operation</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Check wrong address parity</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Early strobe margin</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Late strobe margin</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>Low current margin</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>High current margin</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Reserved</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>Reserved</td></tr>
</table>
All of Main Memory is margined simultaneously.</td></tr>
</table>

of memory is present. The Low and High Error Address Registers indicate the 32K section of memory in which the error occurred. The Error Register indicates whether the error occurred on the odd or even addressed word. If, for instance, the error occurred in the odd addressed word, the 16K section containing odd addressed words should be replaced.

If an FDM parity error bit is set in the Error Register, the bad chip is on the M8144 (DTM) module. Knowing which group failed and the state of address bit A01 (from the Low Error Address Register), it can be determined which of the four word sections of the FDM (Group 0 even and odd, Group 1 even and odd) has failed.

If an Address Memory parity bit is set in the Error Register, the problem is on the M8143 (ADM) module. The Error Register indicates whether the error occurred in the Tag 0 or Tag 1 Address Memory.

If the Main Memory Address Parity bit is set, there may be a problem in the parity generator (Drawing ADMJ) or in a memory controller parity checker. A failure in the Main Memory Bus address and control lines is the most likely cause for this error.

If the Main Memory time-out bit is set, the most probable cause is a memory controller failure. Another possible cause is a misconfiguration of the System Size Register in the processor.

4.8.8 Register Logic

The Cache device registers and their associated logic are located on Drawings CCBF, H, J, K, and L. Figure 4-14 is a block diagram showing the Cache device registers and associated logic. Each block in the figure references the page of the engineering schematics on which the logic is located.

Read Multiplexer – The Read Multiplexer gates the contents of one of the Cache registers onto the REG D15–00 H lines. Register bits 15, 14, and 05:00 are multiplexed by 8-line to 1-line multiplexers on Drawing CCBF. These multiplexers are controlled by a decode of Unibus address bits MAPA ADRS 03:01 H (gated by the Unibus Map).

The remaining register bits are multiplexed by dual 4:1 line multiplexers shown on Drawing CCBF. These multiplexers are controlled by an independent decode of the Unibus address bits.

Register Write Select Logic – The Register Write Select Logic consists of a BCD to one of ten decoder (type 7442) and some gating. The A, B, and C inputs of the decoder are bits (03:01) of the Unibus address. When input D of the decoder goes low, one of the three writable Cache registers may be written. Input D goes low at T60 of the Cache timing sequence when the Cache is performing a Unibus Map write operation during which MAPB CACHE REG H is asserted. If the A, B, and C inputs indicate a binary 2, 3, or 4, the "2," "3," or "4" output of the decoder goes low when input D goes low; this causes the assertion of CCBH CLK CONTROL REG L, CCBH CLK MAINT REGL,or CCBH WRITE ERR REG L.

CCBH CLK CONTROL REG L clocks the Control Register and loads it with the data gated from the Unibus. CCBH CLK MAINT REG L clocks the Maintenance Register (Drawing CCBL) and loads it with the data gated from the Unibus.

CCBH WRITE ERR REG L is input to a set of four type 8266 multiplexers (Drawings CCBJ and CCBK), selecting the data gated from the Unibus. This data is inverted and applied to the clear inputs of the Error Register flip-flops. Thus a 1 bit is inverted to a low level which clears the corresponding Error Register bit.

Trap and Abort Logic – The Cache asserts CCBJ PARITY TRAP H when one of the two trap request flip-flops is set. One of the flip-flops is set when the Unibus Map asserts PB on the Unibus (MAPB PB DATA H asserted). The other flip-flop is set when CCBK ANY ERR (1) H is asserted during a valid processor cycle (CCBJ VALID CP CYC H asserted) or a Unibus Map memory (i.e., non-register) cycle.

CCBK ANY ERR (1) H is asserted as a result of:

1. A time-out on the Main Memory Bus during a non-MBC cycle.

2. A parity error on data read from the FDM.

3. A parity error on address tags read from the Address Memory.

4. A parity error on data read from Main Memory during a non-MBC cycle.

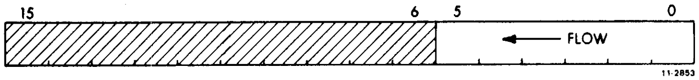


Figure 4-13 Hit/Miss Register

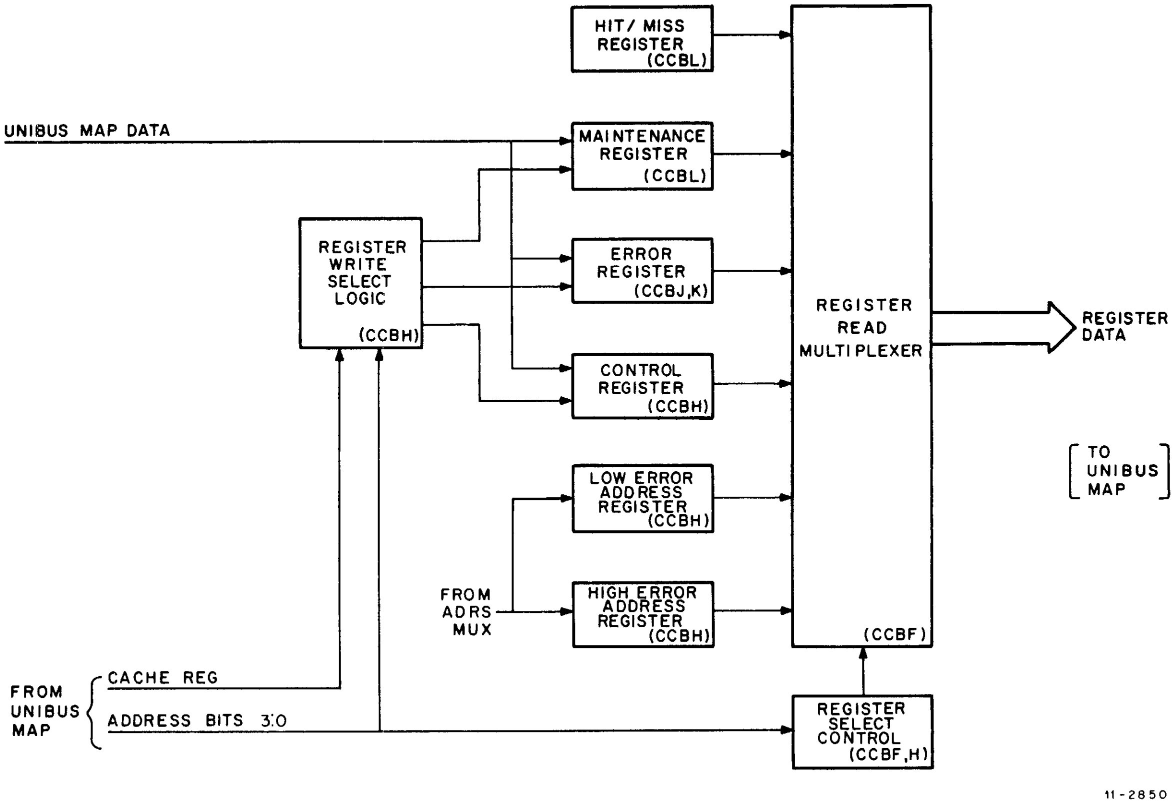

HIT/ MISS REGISTER (CCBL)
UNIBUS MAP DATA
MAINTENANCE REGISTER (CCBL)
REGISTER WRITE SELECT LOGIC (CCBH)
ERROR REGISTER (CCBJ,K)
CONTROL REGISTER (CCBH)
REGISTER READ MULTIPLEXER (CCBF)
REGISTER DATA
TO UNIBUS MAP
LOW ERROR ADDRESS REGISTER (CCBH)
FROM ADRS MUX
HIGH ERROR ADDRESS REGISTER (CCBH)
FROM UNIBUS MAP
CACHE REG
ADDRESS BITS 3:0
REGISTER SELECT CONTROL (CCBF,H)
11-2850

Figure 4-14 Register Logic Block Diagram

If traps are disabled (CCBH DIS TRAPS L and CCBH DIS UNI TRAPS L asserted), assertion of CCBH PARITY TRAP H is inhibited.

The trap request flip-flops are cleared upon initialization (CCBA INIT D L asserted) or when the processor acknowledges receipt of a trap request (TMCA PERF ACKN L asserted) or abort acknowledge (PDRH CACHE PERF L asserted). The trap request flip-flops are also cleared when the processor traps due to a Unibus parity error (UBCB UBUS PAR ERR H asserted); this is done because the Unibus parity error trap routine will also handle other concurrent trap conditions.

The Cache asserts CCBJ PARITY ABORT H to abort the processor. This occurs if the Cache cannot supply good data (DTMM BAD PARITY H asserted) to the processor, or when a Main Memory Bus timeout occurs during a processor cycle (CCBD CP TIMEOUT L asserted).

The processor acknowledges receipt of CCBJ PARITY ABORT H by asserting PDRH CACHE PERF L. In an abort due to a timeout, the processor asserts PDRH CACHE PERF L in response to CCBD CP TIMEOUT L; PDRH CACHE PERF L then asserts CCBJ PARITY ABORT H.

Error Address Register Logic – The Error Address Register (Drawing CCBH) is in an undetermined state at power-up, and is loaded with a 22-bit physical address and operation control bits at T60 of each Cache cycle. If any error is detected, further clocking of this register is inhibited by the negation of CCBJ CLK ADRS H. Thus, the address at which the error occurred is maintained in the Error Address Register.

CCBJ CLK ADRS H is negated when one of the trap request flip-flops is set.

Clocking of the Error Address Register may again be enabled by servicing the error condition that caused CCBJ CLK ADRS H to be negated. Note that CCBJ CLK ADRS H can be immediately asserted by simultaneously clearing Error Register bits 15 and 13.

Negation of CCBJ CLK ADRS H causes the assertion of CCBJ AOK (0) H. Therefore, if another error occurs after the error that caused the Error Address Register to lock, bit 14 or 12 of the Error Register is set.

APPENDIX A

BLOCK DIAGRAMS

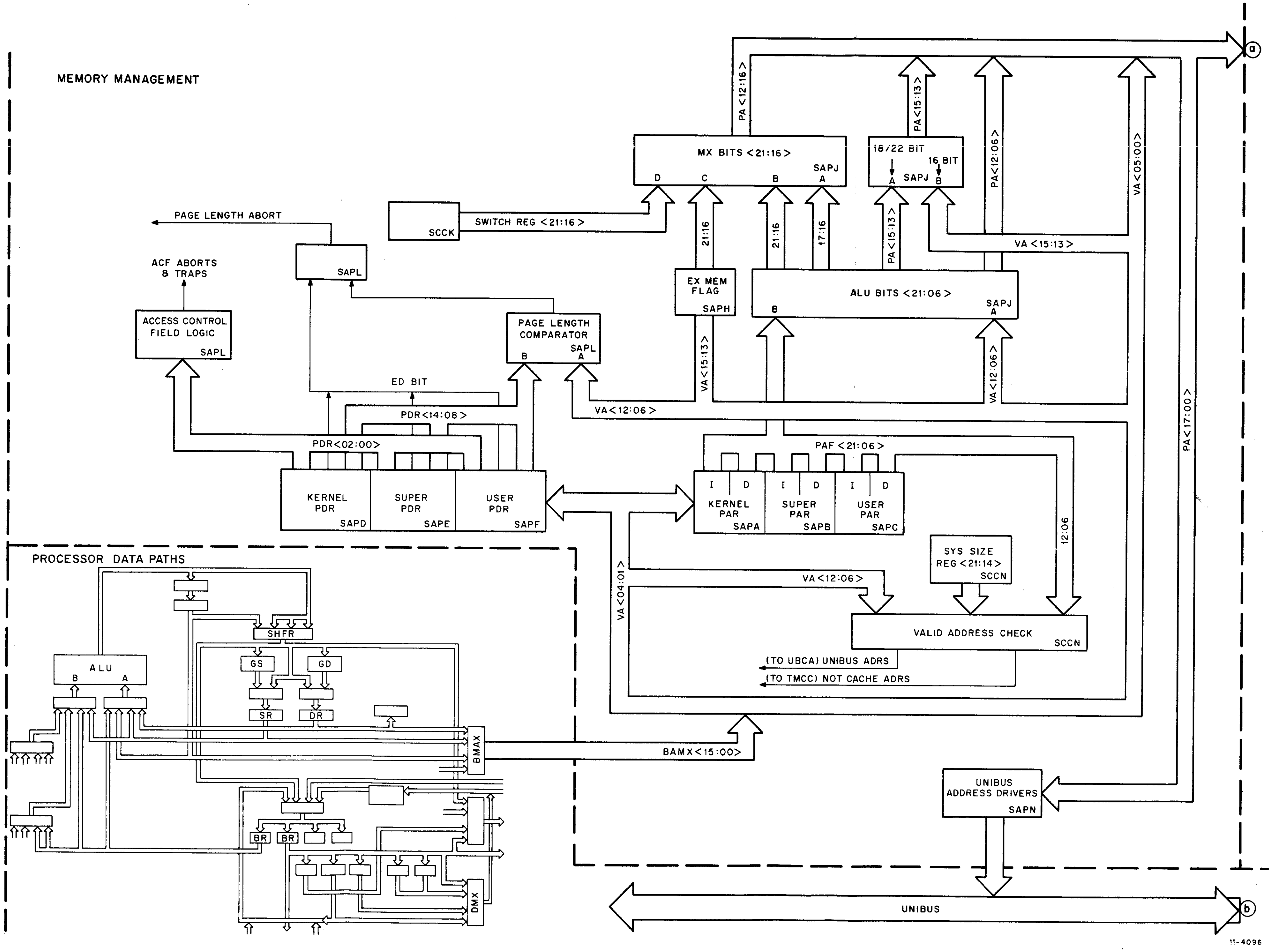


Figure A-1 PDP-11/70 Address Paths Block Diagram (Sheet 1 of 2)

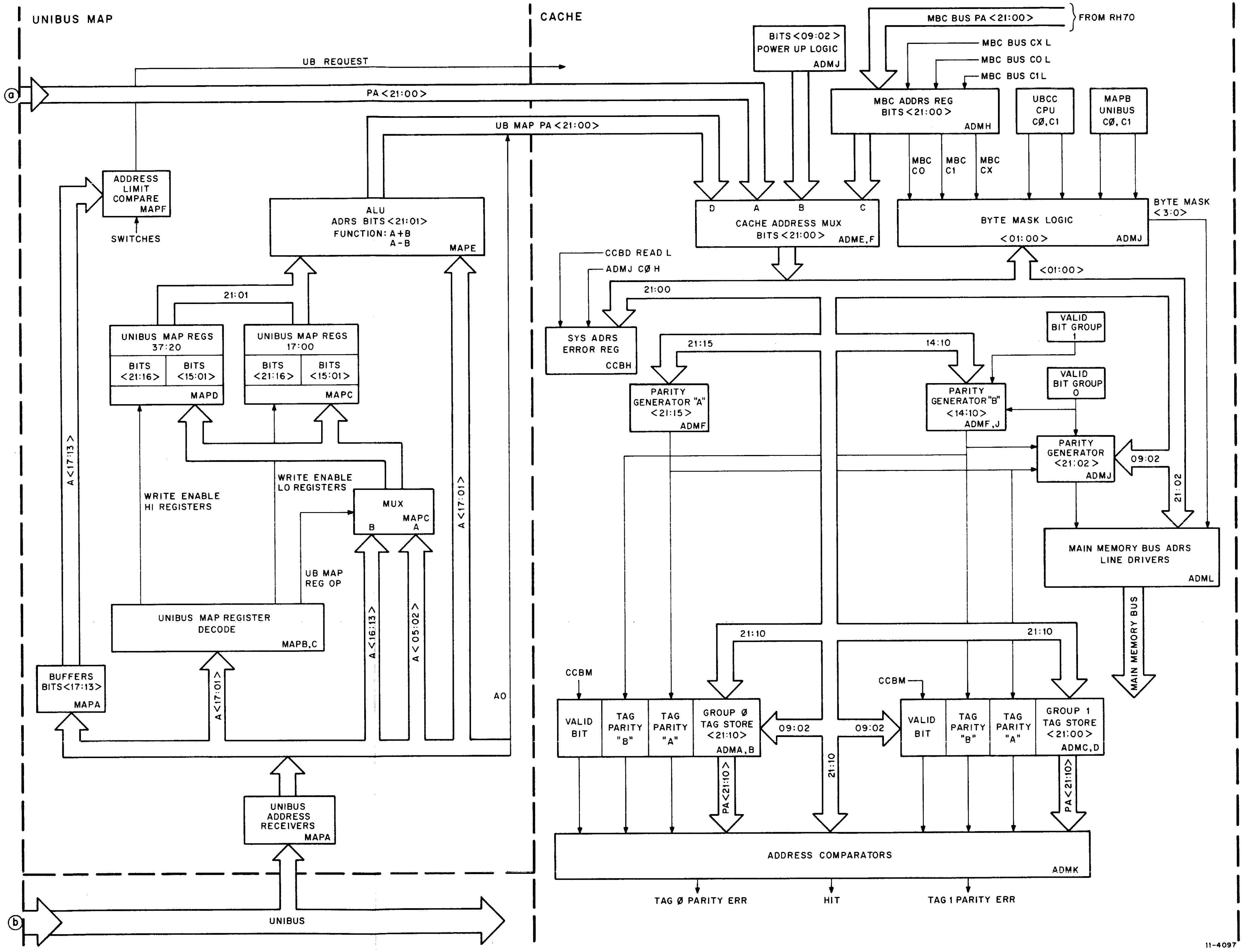


Figure A-1 PDP-11/70 Address Paths
Block Diagram (Sheet 2 of 2)

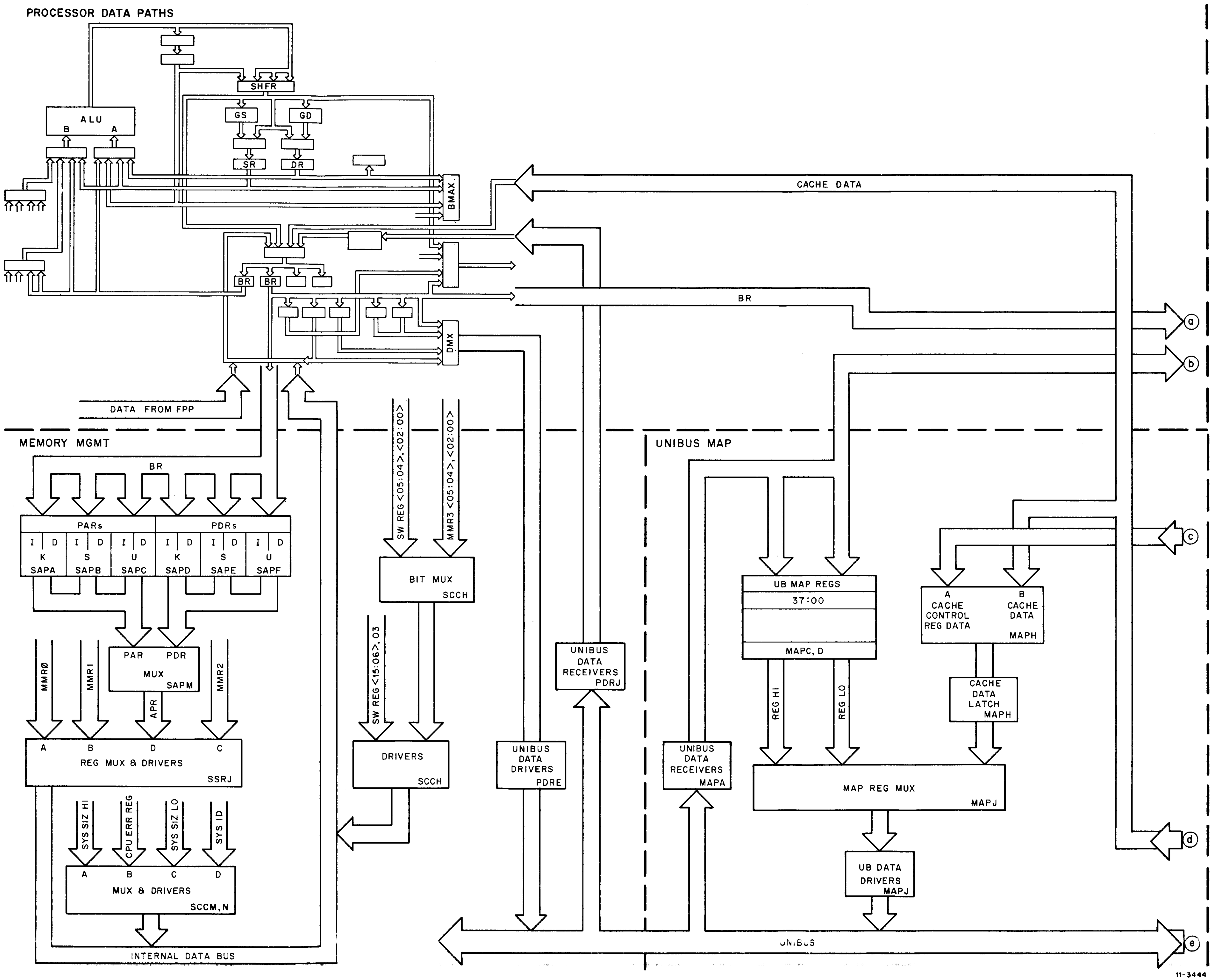


Figure A-2 PDP-11/70 Data Paths Block Diagram (Sheet 1 of 2)

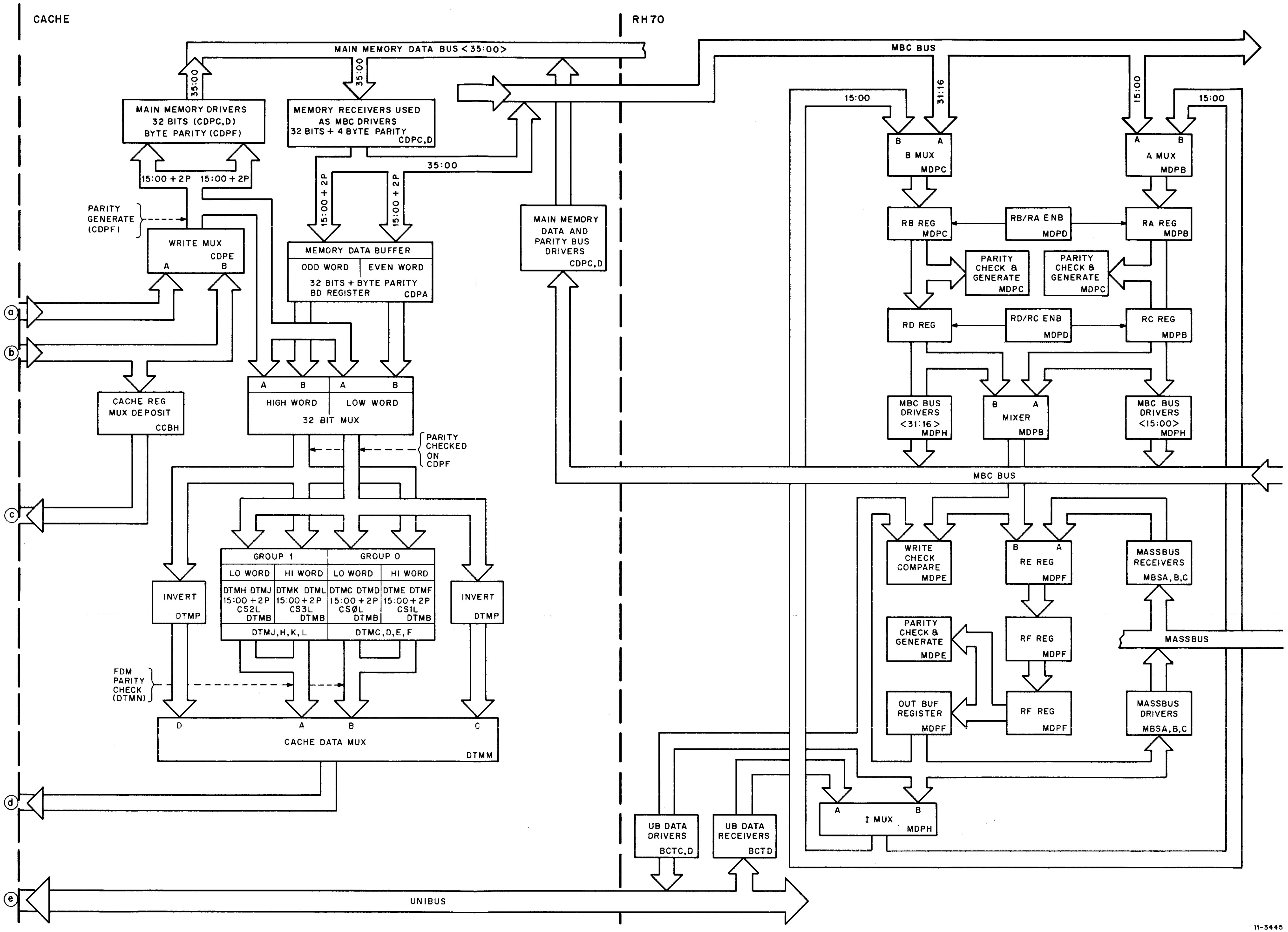


Figure A-2 PDP-11/70 Data Paths Block Diagram (Sheet 2 of 2)

INDEX

This Index lists the principal references to the CPU modules (slots 6 through 22 of the processor back plane). Each module schematic sheet is listed separately. Roman numerals indicate the *Section* of this manual which contains the reference; arabic numerals indicate the Chapter and the Paragraph within the Section.

KB11-C PROCESSOR (PDP-11/70)
EK-KB11C-TM-001

Reader's Comments

Your comments and suggestions will help us in our continuous effort to improve the quality and usefulness of our publications.

What is your general reaction to this manual? In your judgment is it complete, accurate, well organized, well written, etc.? Is it easy to use? ______________________________

What features are most useful? ______________________________

What faults do you find with the manual? ______________________________

Does this manual satisfy the need you think it was intended to satisfy? ______________

Does it satisfy *your* needs? ______________ Why? ______________

Would you please indicate any factual errors you have found. ______________

Please describe your position. ______________________________

Name ______________ Organization ______________

Street ______________ Department ______________

City ______________ State ______________ Zip or Country ______________

CUT OUT ON DOTTED LINE

— — — — — — — — — — — Fold Here — — — — — — — — — — — —

— — — — — — — — — Do Not Tear - Fold Here and Staple — — — — — — — — —

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

Digital Equipment Corporation
Technical Documentation Department
146 Main Street
Maynard, Massachusetts 01754