1977-78
digital
pdp11/70
processor
handbook

digital

DIGITAL EQUIPMENT CORPORATION, Corporate Headquarters: Maynard, Massachusetts 01754, Telephone: (617)897-5111—SALES AND SERVICE OFFICES: UNITED STATES—ALABAMA, Huntsville • ARIZONA, Phoenix and Tucson • CALIFORNIA, El Segundo, Los Angeles, Oakland, Ridgecrest, San Diego, San Francisco (Mountain View), Santa Ana, Santa Clara, Stanford, Sunnyvale and Woodland Hills • COLORADO, Englewood • CONNECTICUT, Fairfield and Meriden • DISTRICT OF COLUMBIA, Washington (Lanham, MD) • FLORIDA, Ft. Lauderdale and Orlando • GEORGIA, Atlanta • HAWAII, Honolulu • ILLINOIS, Chicago (Rolling Meadows) • INDIANA, Indianapolis • IOWA, Bettendorf • KENTUCKY, Louisville • LOUISIANA, New Orleans (Metairie) • MARYLAND, Odenton • MASSACHUSETTS, Marlborough, Waltham and Westfield • MICHIGAN, Detroit (Farmington Hills) • MINNESOTA, Minneapolis • MISSOURI, Kansas City (Independence) and St. Louis • NEW HAMPSHIRE, Manchester • NEW JERSEY, Cherry Hill, Fairfield, Metuchen and Princeton • NEW MEXICO, Albuquerque • NEW YORK, Albany, Buffalo (Cheektowaga), Long Island (Huntington Station), Manhattan, Rochester and Syracuse • NORTH CAROLINA, Durham/Chapel Hill • OHIO, Cleveland (Euclid), Columbus and Dayton • OKLAHOMA, Tulsa • OREGON, Eugene and Portland • PENNSYLVANIA, Allentown, Philadelphia (Bluebell) and Pittsburgh • SOUTH CAROLINA, Columbia • TENNESSEE, Knoxville and Nashville • TEXAS, Austin, Dallas and Houston • UTAH, Salt Lake City • VIRGINIA, Richmond • WASHINGTON, Bellevue • WISCONSIN, Milwaukee (Brookfield) • INTERNATIONAL—ARGENTINA, Buenos Aires • AUSTRALIA, Adelaide, Brisbane, Canberra, Melbourne, Perth and Sydney • AUSTRIA, Vienna • BELGIUM, Brussels • BOLIVIA, La Paz • BRAZIL, Rio de Janeiro and Sao Paulo • CANADA, Calgary, Edmonton, Halifax, London, Montreal, Ottawa, Toronto, Vancouver and Winnipeg • CHILE, Santiago • DENMARK, Copenhagen • FINLAND, Helsinki • FRANCE, Lyon, Grenoble and Paris • GERMAN FEDERAL REPUBLIC, Cologne, Frankfurt, Hamburg, Hannover, Munich, Nuremburg, Stuttgart and West Berlin • HONG KONG • INDIA, Bombay • INDONESIA, Djakarta • IRELAND, Dublin • ITALY, ITALY, Milan, Rome and Turin • IRAN, Tehran • JAPAN, Osaka and Tokyo • MALAYSIA, Kuala Lumpur • MEXICO, Mexico City • NETHERLANDS, Utrecht • NEW ZEALAND, Auckland and Christchurch • NORWAY, Oslo • PUERTO RICO, Santurce • SINGAPORE • SPAIN, Madrid • SWEDEN, Gothenburg and Stockholm • SWITZERLAND, Geneva and Zurich • UNITED KINGDOM, Birmingham, Bristol, Epsom, Edinburgh, Leeds, Leicester, London, Manchester and Reading • VENEZUELA, Caracas •

digital

pdp11/70

processor handbook

digital equipment corporation

Copyright © 1976 by Digital Equipment Corporation

DEC, PDP, UNIBUS are registered trademarks
of Digital Equipment Corporation.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION 1-1
1.1 PDP-11/70 1-1
1.2 FEATURES 1-1
1.3 SYSTEM ARCHITECTURE 1-1
1.4 CENTRAL PROCESSOR 1-2
1.5 MEMORY 1-6
1.6 MEMORY SYSTEM 1-7
1.7 OTHER CPU EQUIPMENT 1-8
1.8 UNIBUS 1-9
1.9 SYSTEM INTERACTION 1-10
1.10 THE PDP-11 FAMILY 1-14
1.11 PERIPHERALS/OPTIONS 1-14

CHAPTER 2 SPECIFICATIONS 2-1
2.1 PACKAGING 2-1
2.2 COMPONENT PARTS 2-1
2.3 OTHER SPECIFICATIONS 2-2

CHAPTER 3 ADDRESSING MODES 3-1
3.1 SINGLE OPERAND ADDRESSING 3-2
3.2 DOUBLE OPERAND ADDRESSING 3-2
3.3 DIRECT ADDRESSING 3-3
3.4 DEFERRED ADDRESSING 3-8
3.5 USE OF PC 3-11
3.6 USE OF STACK POINTER 3-14

CHAPTER 4 INSTRUCTION SET 4-1
4.1 INTRODUCTION 4-1
4.2 INSTRUCTION FORMATS 4-2
4.3 BYTE INSTRUCTIONS 4-3
4.4 SINGLE OPERAND INSTRUCTIONS 4-5
4.5 DOUBLE OPERAND INSTRUCTIONS 4-25
4.6 PROGRAM CONTROL INSTRUCTIONS 4-37
4.7 MISCELLANEOUS INSTRUCTIONS 4-78
4.8 CONDITION CODE OPERATORS 4-86

CHAPTER 5 PROCESSOR CONTROL 5-1
5.1 GENERAL 5-1
5.2 REGISTERS 5-1
5.3 PROCESSOR TRAPS 5-2
5.4 STACK LIMIT 5-4
5.5 PROGRAM INTERRUPT REQUESTS 5-5

CHAPTER 6 ADDRESSING 6-1
6.1 GENERAL 6-1
6.2 ADDRESS SPACE 6-1
6.3 CPU MAPPING 6-2
6.4 COMPATIBILITY 6-2

6.5 MEMORY MANAGEMENT 6-4
6.6 UNIBUS MAP 6-20
6.7 NON-EXISTENT MEMORY ERRORS 6-21

CHAPTER 7 MEMORY SYSTEM 7-1
7.1 GENERAL 7-1
7.2 CACHE MEMORY 7-1
7.3 PARITY 7-5
7.4 REGISTERS 7-7

CHAPTER 8 FLOATING POINT PROCESSOR 8-1
8.1 INTRODUCTION 8-1
8.2 OPERATION 8-1
8.3 ARCHITECTURE 8-2
8.4 FLOATING POINT DATA FORMATS 8-3
8.5 FPP STATUS REGISTER 8-5
8.6 FEC REGISTER 8-9
8.7 FPP INSTRUCTION ADDRESSING 8-9
8.8 FPP INSTRUCTIONS 8-11

CHAPTER 9 PROGRAMMING TECHNIQUES 9-1
9.1 THE STACK 9-1
9.2 SUBROUTINE LINKAGE 9-5
9.3 INTERRUPTS 9-9
9.4 REENTRANCY 9-11
9.5 POSITION INDEPENDENT CODE 9-14
9.6 CO-ROUTINES 9-15

CHAPTER 10 HIGH-SPEED I/O CONTROLLERS 10-1
10.1 SYSTEM PERFORMANCE 10-1
10.2 HIGH-SPEED, MASS STORAGE PERIPHERALS 10-1
10.3 HIGH-SPEED CONTROLLERS 10-3
10.4 REGISTERS 10-4
10.5 CONTROLLER REGISTERS 10-5

CHAPTER 11 CONSOLE OPERATION 11-1
11.1 INTRODUCTION 11-1
11.2 GENERAL 11-1
11.3 STARTING AND STOPPING 11-2
11.4 REFERENCING MEMORY 11-2
11.5 STEP OPERATIONS 11-3
11.6 GENERAL REGISTERS 11-4
11.7 SINGLE INSTRUCTION/SINGLE BUS CYCLE 11-5
11.8 FUNCTIONS OF SWITCHES & INDICATORS 11-5

Appendix A Memory Map A-1

Appendix B Summary of Registers B-1

Appendix C Instruction Timing C-1

CHAPTER 1

INTRODUCTION

1.1 PDP-11/70

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 be 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. It is the systems level PDP-11 that applies the power of 32-bit hardware architecture to demanding, multi-function computing requirements.

1.2 FEATURES

The PDP-11/70 contains as an integral part of the central processor unit, the following hardware features and expansion capabilities:

- Cache memory organization to provide very fast program execution speed and high system throughput
- Memory management for relocation and protection in multi-user, multi-task environments
- Ability to access up to 2 million bytes of main memory (1 byte = 8 bits)
- Optional high-speed, mass storage controllers as an integral part of the CPU, to provide dedicated paths to high performance storage devices
- Optional Floating Point processor with advanced features and operation with 32-bit and 64-bit numbers

1.3 SYSTEM ARCHITECTURE

The PDP-11/70 is a medium scale general purpose computer using an enhanced, upwards-compatible version of the basic architecture of the PDP-11. A block diagram of the computer is shown in Figure 1-1.

The Central 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 a UNIBUS Map which translates UNIBUS addresses to physical memory addresses. The Cache contains 2,048 bytes of fast, bipolar memory that buffers the data from main (core) memory.

Also within the CPU assembly are pre-wired areas for a Floating Point Processor, and up to 4 High-Speed I/O Controllers.

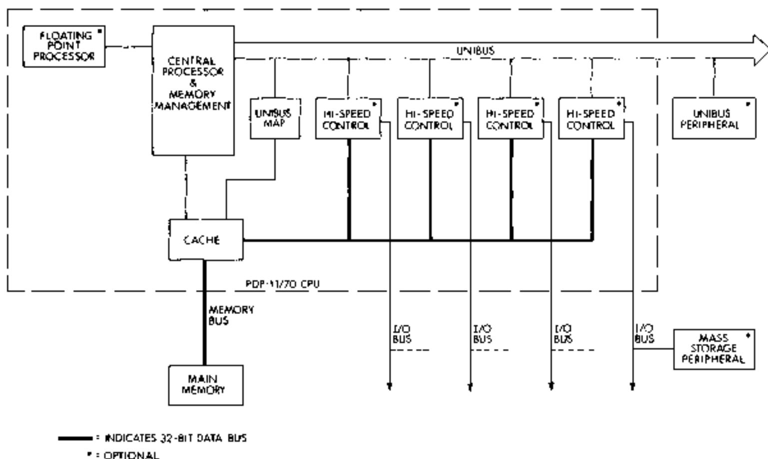


Figure 1-1 PDP-11/70 Block Diagram

The PDP-11/70 System has an expanded internal implementation of the PDP-11 architecture for greatly improved systems thruput. All the memory is on its own high data rate bus. The internal high-speed I/O controllers for mass storage devices have direct connections through the cache to memory for transferring data (using the cache only for timing purposes). The processor has a direct connection to the cache memory system for very high-speed memory access.

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 absolutely no effect on programming the PDP-11/70.

1.4 CENTRAL PROCESSOR

The PDP-11/70 performs all arithmetic and logical operations required in the system. 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 central 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. It also provides room for the addition of the high-speed Floating Point Processor, and High-Speed Controllers.

The machine operates in three modes: Kernel, Supervisor, and User. When the machine is in Kernel mode a program has complete control of the machine; when the machine is in any other mode the processor is inhibited from executing certain instructions and can be denied direct access to the perpiherals on the system. This hardware feature can be used to provide complete executive protection in a multi-programming environment.

The central processor contains 16 general registers which can be used as accumulators, index registers, or as stack pointers. Stacks are extremely useful for nesting programs, creating re-entrant coding, and as temporary storage where a Last-In First-Out structure is desirable. One of the general registers is used as the PDP-11/70's program counter. Three others are used as Processor Stack Pointers, one for each operational mode.

The CPU performs all of the computer's computation and logic operations in a parallel binary mode through step by step execution of individual instructions.

1.4.1 General Registers

The general registers can be used for a variety of purposes; the uses varying with requirements. The general registers can be used as accumulators, index registers, autoincrement registers, autodecrement registers, or as stack pointers for temporary storage of data. Chapter 3 on Addressing describes these uses of the general registers in more detail. Arithmetic operations can be from one general register to another, from one memory or device register to another, or between memory or a device register and a general register.

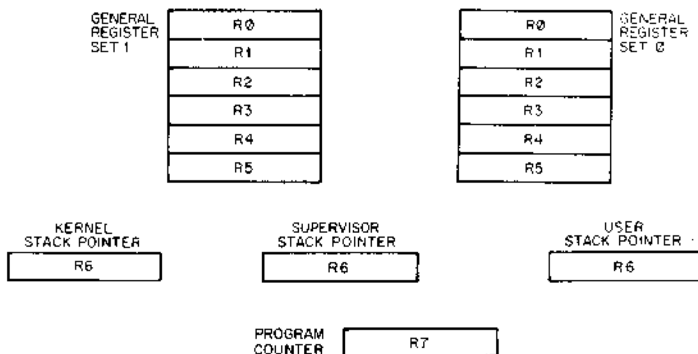


Figure 1-2 The General Registers

R7 is used as the machine's program counter (PC) and contains the address of the next instruction to be executed. It is a general register normally used only for addressing purposes and not as an accumulator for arithmetic operations.

The R6 register is normally used as the Processor Stack Pointer indicat-

ing the last entry in the appropriate stack (a common temporary storage area with "Last-In First-Out" characteristics). (For information on the programming uses of stacks, please refer to Chapter 9). The three stacks are called the Kernel Stack, the Supervisor Stack, and the User Stack. When the Central Processor is operating in Kernel mode it uses the Kernel Stack, in Supervisor mode, the Supervisor Stack, and in User mode, the User Stack. When an interrupt or trap occurs, the PDP-11/70 automatically saves its current status on the Processor Stack selected by the service routine. This stack-based architecture facilitates reentrant programming.

The remaining 12 registers are divided into two sets of unrestricted registers, R0-R5. The current register set in operation is determined by the Processor Status Word.

The two sets of registers can be used to increase the speed of real-time data handling or facilitate multi-programming. The six registers in General Register Set 0 could each be used as an accumulator and/or index register for a real-time task or device, or as general registers for a Kernel or Supervisor mode program. General Register Set 1 could be used by the remaining programs or User mode programs. The Supervisor can therefore protect its general registers and stack from User programs, or other parts of the Supervisor.

1.4.2 Processor Status Word

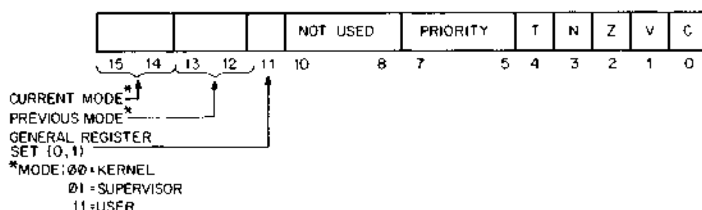


Figure 1-3 Processor Status Word

The Processor Status Word, located at location 17777776, contains information on the current status of the PDP-11/70. This information includes the register set currently in use; current processor priority; current and previous operational modes; the condition codes describing the results of the last instruction; and an indicator for detecting the execution of an instruction to be trapped during program debugging.

Modes

Mode information includes the present mode, either User, Supervisor, or Kernel (bits 15, 14); the mode the machine was in prior to the last interrupt or trap (bits 13, 12); and which register set (General Register Set 0 or 1) is currently being used (bit 11).

The three modes permit a fully protected environment for a multi-programming system by providing the user with three distinct sets of Processor Stacks and Memory Management Registers for memory mapping.

In all modes except Kernel a program is inhibited from executing a "HALT" instruction and the processor will trap through location 4 if an attempt is made to execute this instruction. Furthermore, the processsor will ignore the "RESET" and "SPL" (Set Priority level) instructions. In Kernel mode, the processor will execute all instructions.

A program operating in Kernel mode can map users' programs anywhere in core and thus explicitly protect key areas (including the devices registers and the Processor Status Word) from the User operating environment.

Processor Priority
The Central Processor operates at any of eight levels of priority, 0-7. When the CPU is operating at level 7 an external device cannot interrupt it with a request for service. The Central Processor must be operating at a lower priority than the priority of the external device's request in order for the interruption to take effect. The current priority is maintained in the processor status word (bits 5-7). The 8 processor levels provide an effective interrupt mask, which can be dynamically altered through use of the Set Priority Level (SPL) instruction which is described in Chapter 4 and which can only be used by the Kernel. This instruction allows a Kernel mode program to alter the Central Processor's priority without affecting the rest of the Processor Status Word.

Condition Codes
The condition codes contain information on the result of the last CPU operation. They include: a carry bit (C), which is set by the previous operation if the operation caused a carry out of its most significant bit; a negative bit (N) set if the result of the previous operation was negative; a zero bit (Z), set if the result of the previous operation was zero; and an overflow bit (V), set if the result of the previous operation resulted in an arithmetic overflow.

Trap
The trap bit (T) can be set or cleared under program control. When set, a processor trap will occur through location 14 on completion of instruction execution and a new Processor Status Word will be loaded. This bit is especially useful for debugging programs as it provides an efficient method of installing breakpoints.

Interrupts and trap instructions both automatically cause the previous Processor Status Word and Program Counter to be saved and replaced by the new values corresponding to those required by the routine servicing the interrupt or trap. The user can, thus, cause the central processor to automatically switch modes (context switching), register sets, alter the CPU's priority, or disable the Trap Bit whenever a trap or interrupt occurs.

1.4.3 Stack Limit Register
All PDP-11's have a Stack Overflow Boundary at location 400. The Kernel Stack Boundary, in the PDP-11/70 is a variable boundary set through the Stack Limit Register found in location 17777774.

Once the Kernel stack exceeds its boundary, the Processor will complete the current instruction and then trap to location 4 (Yellow or Warning Stack Violation). If, for some reason, the program persists beyond the

16-word limit, the processor will abort the offending instruction, set the stack pointer (R6) to 4 and trap to location 4 (Red or Fatal Stack Violation). A description of these traps is contained in Appendix A.

1.5 MEMORY

Memory Organization
A memory can be viewed as a series of locations, with a number (address) assigned to each location. Thus a 16,384-byte PDP-11 memory could be shown as in Figure 2-5.

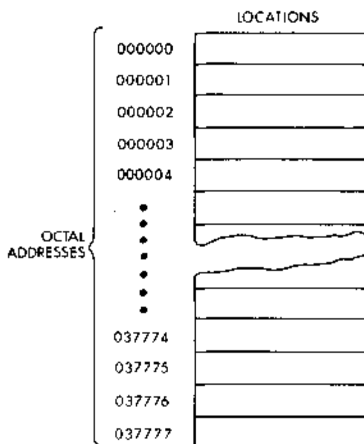


Figure 1-4 Memory Addresses

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. An 8K-word memory can contain 16K bytes and consist of 037777 octal locations. Words always start at even-numbered locations.

A PDP-11 word is divided into a high byte and a low byte as shown in Figure 1-5.

Figure 1-5 High & Low Byte

Low bytes are stored at even-numbered memory locations and high bytes at odd-numbered memory locations. Thus it is convenient to view the PDP-11 memory as shown in Figure 1-6.

Certain memory locations have been reserved by the system for interrupt and trap handling, processor stacks, general registers, and periph-

eral device registers. Addresses from 0 to 370_8 are always reserved and those to 777_8 are reserved on large system configurations for traps and interrupt handling.

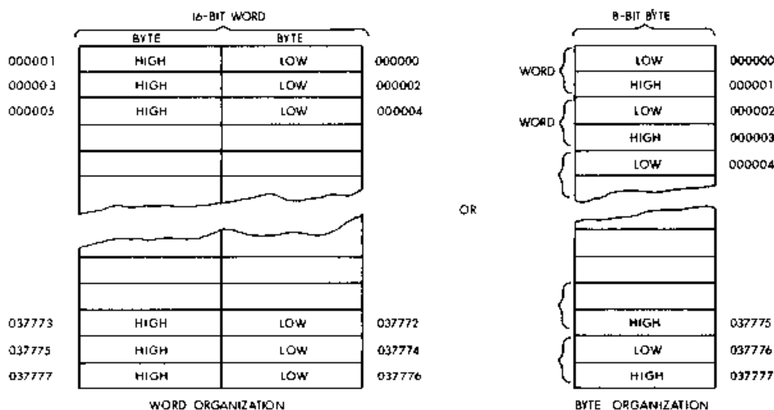


Figure 1-6 Word and Byte Addresses

Parity

Parity is used extensively in the PDP-11/70 to ensure the integrity of information. All memory has byte parity. Parity for both data and addresses is generated on transfers to memory and is checked on all transfers from memory. Registers are provided within the CPU to provide information on the location of parity errors, types of errors, and other relevant information so that software can respond to the situation, take corrective action, and log the occurrence of errors.

1.6 MEMORY SYSTEM

1.6.1 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.

Of the over 4 million byte locations possible with the 22-bit address, the top 256K are used to reference the UNIBUS rarther than physical memory. Maximum main memory is therefore $2^{22} - 2^{18}$, or a total of 3,932,160 bytes, although only 2 million bytes are allowed due to bus length limitations.

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. The information is summarized below:

		Bytes
16 bits	program virtual space	$2^{16} = 64K$
18 bits	UNIBUS space	$2^{18} = 256K$
22 bits	physical memory space	4 million

1.6.2 Memory Management
The Memory Management hardware is standard with the PDP-11/70 computer. It is a hardware relocation and protection facility that can convert the 16-bit program virtual addresses to 22-bit addresses. The unit may be enabled or disabled under program control. There is no increase in access time when the Memory Management unit is enabled.

1.6.3 UNIBUS Map
The UNIBUS Map responds like memory on the UNIBUS. It is the hardware relocation facility for converting the 18-bit UNIBUS addresses to 22-bit addresses. The relocation mapping may be enabled or disabled under program control.

1.6.4 Cache
The cache memory is a very 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 to fetch data from memory, the cache circuitry checks 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 CPU.

When a request is made to write data into memory, it is written both to the cache and to main memory, assuring that main memory is always updated immediately.

The key to the effectiveness of PDP-11/70's cache memory is its size. Because it holds 2,048 bytes at any given point in time, and because programs tend to use localized sections of code and data, the PDP-11/70 cache already contains the next needed data word a very high percentage of the time.

A detailed description of cache memory and the other parts of memory are contained in Chapter 7.

1.7 OTHER CPU EQUIPMENT

1.7.1 Floating Point Processor
The PDP-11/70 Floating Point Processor fits integrally into the Central Processor. 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. There is a detailed description in Chapter 7.

1.7.2 High Speed Mass Storage
The PDP-11/70 bussing structure is optimized for high-speed device transfers. Up to four such devices can be plugged directly into the proc-

essor with a dedicated 32-bit bus feeding through to the core memory. 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 1,024K bytes and a transfer rate of 1 microseconds per byte (1.2 microseconds at 50 Hz). The RS03 is a fixed head disk, 512K bytes, 2 μsec per byte. The TU16 is an industry standard 1,600 bpi tape unit.

1.8 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.

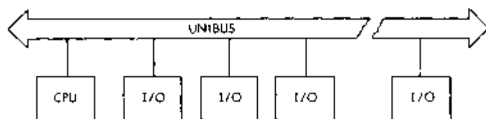


Figure 1-7 PDP-11 System Simplified Block Diagram

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 core memory by the central processor. All the instructions that can be applied to data in core memory can be applied equally well to data in peripheral device registers. This is an especially powerful feature, considering the special capability of PDP-11 instructions to process data in any memory location as though it were an accumulator.

1.8.1 Bidirectional Lines

With bidirectional and asynchronous communications on the UNIBUS, devices can send, receive, and exchange data independently without processor intervention. For example, a cathode ray tube (CRT) display can refresh itself from a disk file while the central processor unit (CPU) attends to other tasks. Because it is asynchronous, the UNIBUS is compatible with devices operating over a wide range of speeds.

1.8.2 Master-Slave Relation

Communication between two devices on the bus is in the form of a master-slave relationship. At any point in time, there is one device that has control of the bus. This controlling device is termed the "bus master." The master device controls the bus when communicating with another device on the bus, termed the "slave." A typical example of this relationship is the processor, as master, fetching an instruction from memory (which is always a slave). Another example is the disk, as master, transferring data to memory, as slave. Master-slave relationships are dynamic. The processor, for example, may pass bus control

to a disk. The disk, as master, could then communicate with a slave memory bank.

Since the UNIBUS is used by the processor and all I/O devices, there is a priority structure to determine which device gets control of the bus. Every device on the UNIBUS which is capable of becoming bus master is assigned a priority. When two devices, which are capable of becoming a bus master, request use of the bus simultaneously, the device with the higher priority will receive control.

1.8.3 Interlocked Communication

Communication on the UNIBUS is interlocked so that for each control signal issued by the master device, there must be a response from the slave in order to complete the transfer. Therefore, communication is independent of the physical bus length (as far as timing is concerned) and the response time of the master and slave devices. The asynchronous operation precludes the need for synchronizing with, and waiting for, clock pulses. Thus, each device is allowed to operate at its maximum possible speed.

Interfaces to the UNIBUS are not time-dependent; there are no pulse-width or rise-time restrictions to worry about.

Input/output devices transferring directly to or from memory are given highest priority and may request bus mastership and steal bus and memory cycles during instruction operations. The processor resumes operation immediately after the memory transfer. Multiple devices can operate simultaneously at maximum direct memory access (DMA) rates by "stealing" bus cycles.

Full 16-bit words or 8-bit bytes of information can be transferred on the bus between a master and a slave. The information can be instructions, addresses, or data. This type of operation occurs when the processor, as master, is fetching instructions, operands, and data from memory, and storing the results into memory after execution of instructions. Direct data transfers occur between a peripheral device control and memory.

1.9 SYSTEM INTERACTION

High-speed NPR devices use separate dedicated busses to the individual high-speed I/O controllers. From the controllers there is a single 4-byte wide bus that interfaces to the cache. The order of priorities in the system are:

1) UNIBUS (via UNIBUS Map)
2) High-speed I/O controllers (A through D)
3) CPU

Control information and lower speed data transfers are carried out through the UNIBUS.

A device will request the UNIBUS for one of two purposes:

To make a non-processor (NPR) transfer of data. (Direct Data Transfers such as DMA), or

To interrupt program execution and force the processor to branch to a service routine.

There are two sources of interrupts, hardware and software.

1.9.1 Hardware Interrupt Requests
A hardware interrupt occurs when a device wishes to indicate to the program, or Central Processor, that a condition has occurred (such as transfer completed, end of tape, etc.). The interrupt can occur on any one of the four Bus Request levels and the processor responds to the interrupt through a service routine.

1.9.2 Program Interrupt Requests
Hardware interrupt servicing is often a two-level process. The first level is directly associated with the device's hardware interrupt and consists of retrieving the data. The second, is a software task that manipulates the raw information. The second process can be run at a lower priority than the first, because the PDP-11/70 provides the user with the means of scheduling his software servicing through seven levels of Program Interrupt Requests. The Program Interrupt Request Register is located at address 17777772. An interrupt is generated by the programmer setting a bit in the high order byte of this register.

1.9.3 Priority Structure on the UNIBUS
When a device capable of becoming bus master requests use of the bus, handling of the request depends on the hierarchical position of that device in the priority structure.

The relative priority of the request is determined by the Processor's priority and the level at which the request is made.

> The processor's priority is set under program control to one of eight levels using bits 7-5 in the processor Status Word. Bus requests are inhibited on the same or lower levels.
>
> Bus requests from external devices can be made on any one of the five request lines. A non-processor request (NPR) has the highest priority, and its request is granted between bus cycles of an instruction execution. But Request 7 (BR 7) is the next highest priority and Bus Request 4 (BR 4) is the lowest. The four lower priority level requests (BR 7-BR 4) are granted by the processor between instructions providing that they occur on higher levels than the processor's. Therefore an interrupt may only occur on a Bus Request Level and not on a Non Processor Request level.
>
> Any number of devices can be connected to a specific BR or NPR line.
>
> If two devices with the same priority request the bus, the device physically closest to the processor on the UNIBUS has the higher priority.
>
> Program Interrupt Requests can be made on any one of 7 levels (PIR 7-PIR 1). Requests are granted by the processor between instructions providing that they occur on higher levels than the processor's.
>
> Program Interrupt Requests take precedence over equivalent level Bus Requests.

1.9.4 Non-Processor Data Transfers
Direct memory or direct data transfers can be accomplished between

any two peripherals without processor supervision. These Non-Processor transfers, called NPR level data transfers, are usually made for Direct Memory Access (memory to/from mass storage) or direct device transfers (disk refreshing a CRT display).

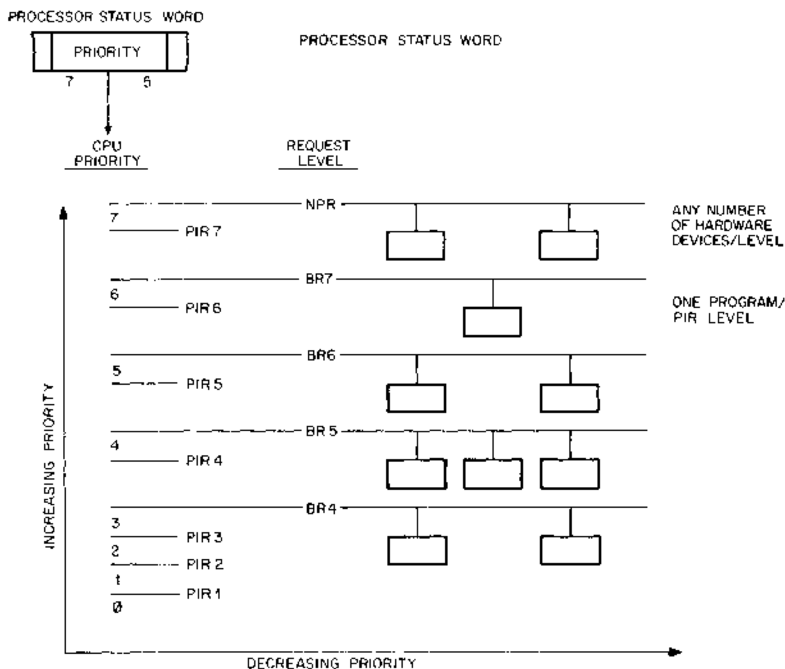


Figure 1-8 UNIBUS Priority Structure

An NPR device provides extremely fast access to the UNIBUS and can transfer data at high rates once it gains control of the bus. The state of the processor is not affected by this type of transfer, and, therefore, the processor can relinquish bus control while an instruction is still in progress. The bus can be released at the end of any bus cycle, except during a read-modify-write cycle sequence. (This occurs for example in destructive read-out devices such as core memory for certain instructions.) In the PDP-11/70 an NPR device can gain bus control in 3.5 microseconds or less (depending on the number of devices on the UNIBUS), and can transfer 16-bit words to memory at the same speed as the effective cycle time of the memory being addressed.

1.9.5 Using the Interrupts

Devices that gain bus control with one of the Bus Request Lines (BR 7-BR 4), can take full advantage of the Central Processor by requesting an interrupt. In this way, the entire instruction set is available for manipulating data and status registers.

When a service routine is to be run, the current task being performed by the central processor is interrupted, and the device service routine is initiated. Once the request has been satisfied, the Processor returns to its former task. Interrupts may also be used to schedule program execution by using the Program Interrupt Request.

1.9.6 Interrupt Procedure

Interrupt handling is automatic in the PDP-11/70. No device polling is required to determine which service routine to execute. The operations required to service an interrupt are as follows:

1. Processor relinquishes control of the bus, priorities permitting.
2. When a master gains control, it sends the processor an interrupt command and a unique memory address which contains the address of the device's service routine in Kernel virtual address space, called the interrupt vector address. Immediately following this pointer address is a word (located at vector address +2) which is to be used as a new Processor Status Word.
3. The processor stores the current Processor Status Word (PS) and the current Program Counter (PC) into CPU temporary registers.
4. The new PC and PS (the interrupt vector) are taken from the specified address. The old PS and PC are then pushed onto the current stack as indicated by bits 15,14 of the new PS and the previous mode in effect is stored in bits 13,12 of the new PS. The service routine is then initiated.

These operations are performed in approximately 2.5 μsec from the time the control processor receives the interrupt command until the time it starts executing the first instruction of the service routine. This time interval assumes no NPR transfer occurred during this time interval.

5. The device service routine can cause the processor to resume the interrupted process by executing the Return from Interrupt (RTI or RTT) instruction, described in Chapter 4, which pops the two top words from the current processor stack and uses them to load the PC and PS registers.

This instruction requires approximately 1.5 μsec providing there is no NPR request.

A device routine can be interrupted by a higher priority bus request any time after the new PC and PS have been loaded. If such an interrupt occurs, the PC and the PS of the service routine are automatically stored in the temporary registers and then pushed onto the new current stack, and the new device routine is initiated.

1.9.7 Interrupt Servicing

Every hardware device capable of interrupting the processor has a unique pair of locations reserved for its interrupt vector. The first word contains the location of the device's service routine, and the second, the Processor Status Word that is to be used by the service routine. Through proper use of the PS, the programmer can switch the operational mode of the processor, alter the General Register Set in use (con-

text switching), and modify the Processor's Priority level to mask out lower level interrupts.

There is one interrupt vector for the Program Interrupt Request. It will generally be necessary in a multi-processing environment to determine which program generated the PIR and where it is located in memory.

1.9.8 Processor Traps

There are a series of errors and programming conditions which will cause the Central Processor to trap to a set of of fixed locations. These include Power Failure, Odd Addressing Errors, Stack Errors, Timeout Errors, Non-Existent Memory Errors, Memory Parity Errors, Memory Management Violations, Floating Point Processor Exception Traps, Use of Reserved Instructions, Use of the T bit in the Processor Status Word, and use of the IOT, EMT, and TRAP instructions.

1.10 THE PDP-11 FAMILY

The PDP-11 family includes several processors, a large number of peripheral devices and options, and extensive software. PDP-11 computers are architecturally similar and hardware and software upwards compatible, although each machine has some of its own characteristics. New PDP-11 systems will be compatible with existing family members. The user can choose the system which is most suitable to his application, but as needs change or grow he can easily add or change hardware.

1.11 PERIPHERAL OPTIONS

Digital Equipment Corporation designs and manufactures many of the peripheral devices offered with PDP-11s. As a designer and manufacturer of peripherals, DIGITAL can offer extremely reliable equipment, lower prices, more choices, and quantity discounts.

Many processor, input/output, memory, bus, and storage options are available. These devices are explained in detail in the PDP-11 Peripherals Handbook.

1.11.1 Input/Output Devices

The LA36 DECwriter, a totally DIGITAL designed and built teleprinter, is the standard PDP-11 system terminal. It has several advantages over standard electromechanical typewriter terminals, including higher speed, fewer mechanical parts and very quiet operation. I/O capabilities can be increased with high-speed paper tape readers—punches, line printers, card readers or alphanumeric display terminals.

PDP-11 I/O devices include:

- DECwriter teleprinter, LA36
- DECterminal alphanumeric display, VT05, VT50
- Teletypes, LT33
- High-speed line printers, LP11, LS11, LV11
- Cassette, TA11
- High-speed paper tape reader punch, PC11
- Card readers, CR11, CD11
- Synchronous and asynchronous communication interfaces

1.11.2 Storage Devices

Storage devices range from convenient, small-reel magnetic tape units to mass storage magnetic tapes and disk memories. A large number of storage devices, in any combination, may be connected to a PDP-11 system. TU56 DECtapes, highly reliable tape units with small tape reels, designed and built by DIGITAL, are ideal for applications with modest storage requirements. Each DECtape provides storage for 144K 16 bit words. For applications which require handling of large volumes of data, DIGITAL offers the industry compatible TU16 Magtape.

Disk storage devices include fixed head disk units and moving-head removable cartridge and disk pack units. PDP-11 storage devices include:

DECtape, TU56

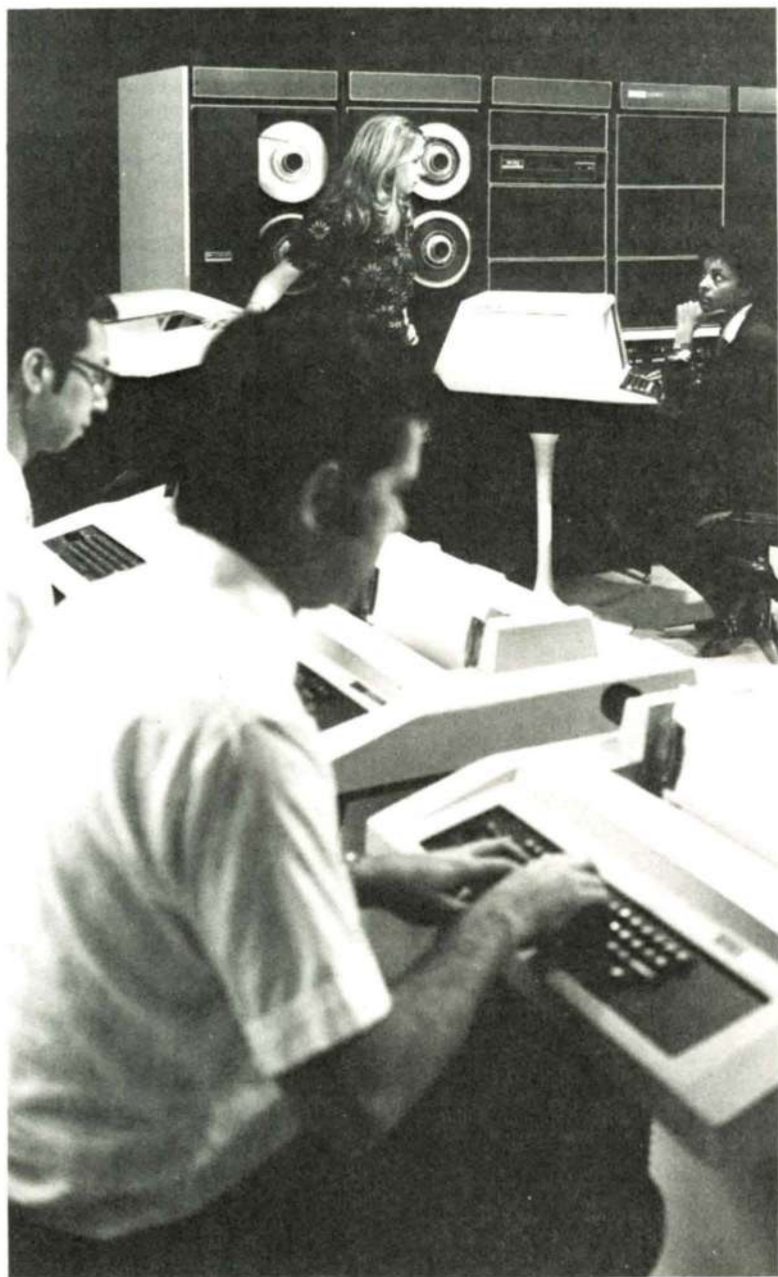
Magtape, TU16

512K byte fixed head disk, RS03

1,024K byte fixed head disk, RS04

2.4M byte moving head cartridge disk, RK05

88M byte moving head disk pack, RP04



CHAPTER 2

SPECIFICATIONS

2.1 PACKAGING

A basic PDP-11/70 consists of two cabinets (see Figure 2-1):

1) A CPU cabinet which contains the processor, CPU related equipment and interface equipment, and
2) A Memory Cabinet which contains the first 128K bytes of parity core memory (with expansion capability to 1,024K bytes within the cabinet. Another memory cabinet located next to it can house an additional 1,024K bytes of memory).

An LA63 DECwriter II console terminal is included with the system. There are prewired areas within the mounting assemblies for expansion with optional equipment.

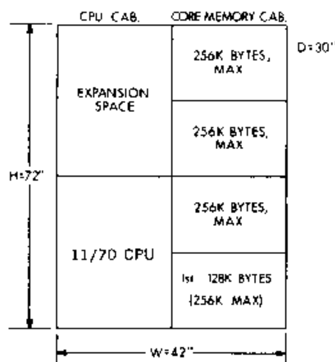


Figure 2-1 Equipment in 11/70 System

2.2 COMPONENT PARTS

The basic PDP-11/70 system has:

Included Equipment

11/70 CPU
Memory Management
Bootstrap loader
Clock (KW11-L)
DECwriter (LA36)
Terminal interface (DL11-A)
2K byte cache memory
128K byte parity core
CPU cabinet
Memory cabinet

Prewired Expansion Space for Optional Equipment
Floating Point Processor
4 High-speed I/O controllers
4 SPC slots for peripherals
128K byte parity core (within 1st memory expansion frame)

2.3 OTHER SPECIFICATIONS

AC Power
115/208 VAC ± 10%, 47 to 63Hz, 3 phase power
230/416 VAC ± 10%, 47 to 63Hz, 3 phase power

	115 VAC	230 VAC
Basic CPU cabinet (current on each of 2 phases):	15A	7.5A
Memory, each 256K bytes (current on 1 phase):	12A	6A

Size
Each cabinet is 72" high x 21" wide x 30" deep.

Weight

CPU cabinet: 500 lbs.

Memory cabinet: 250 lbs. (including 1st 256K bytes)

Memory expansion frame: 150 lbs (each additional 256K bytes)

Operating Environment

Temperature: 10°C to 40°C (50°F to 104°F)

Humidity: 10% to 90% with max wet bulb 28°C (82°F) and minimum dew point 2°C (36°F)

Altitude: to 2.4 km. (8000 ft.)

Non-Operating Environment

Temperature: −40°C to 66°C (−40°F to 151°F)

Humidity: to 95%

Altitude: to 9.1 km (30,000 ft)

CHAPTER 3

ADDRESSING MODES

Data stored in memory must be accessed, and manipulated. Data handling is specified by a PDP-11 instruction (MOV, ADD etc.) which usually indicates:

the function (operation code);

a general purpose register to be used when locating the source operand and/or a general purpose register to be used when locating the destination operand;

an addressing mode (to specify how the selected register(s) is/are to be used.

Since a large portion of the data handled by a computer is usually structured (in character strings, in arrays, in lists etc.), the PDP-11 has been designed to handle structured data efficiently and flexibly. The general registers may be used with an instruction in any of the following ways:

as accumulators. The data to be manipulated resides within the register.

as pointers. The contents of the register are the address of the operand, rather than the operand itself.

as pointers which automatically step through core locations. Automatically stepping forward through consecutive core locations is known as autoincrement addressing; automatically stepping backwards is known as autodecrement addressing. These modes are particularly useful for processing tabular data.

as index registers. In this instance the contents of the register, and the word following the instruction are summed to produce the address of the operand. This allows easy access to variable entries in a list.

PDP-11's also have instruction addressing mode combinations which facilitate temporary data storage structures for convenient handling of data which must be frequently accessed. This is known as the "stack." (see Chapter 9)

In the PDP-11 any register can be used as a "stack pointer" under program control; however, certain instructions associated with subroutine linkage and interrupt service automatically use Register 6 as a "hardware stack pointer." For this reason R6 is frequently referred to as the "SP."

R7 is used by the processor as its program counter (PC). It is recommended that R7 not be used as a stack pointer.

An important PDP-11/70 feature, which must be considered in conjunction with the addressing modes, is the register arrangement;

Two sets of general purpose registers (R0-R5)

three hardware stack pointers (R6)

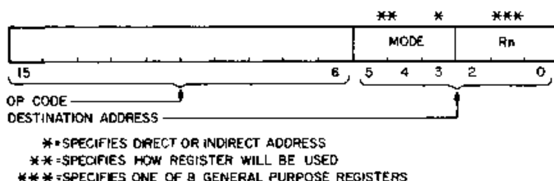
a Program Counter (PC) register (R7).

Register R7 is used as a common program counter (PC). At any point in time only one register set is active. Thus a programmer need only concern himself with the existence of multiple register sets for those special supervisory tasks which involve Kernel, Supervisor, User communications (e.g. MTPX, MFPX); otherwise he need never worry about which R3 or R6 an instruction will reference, the choice is automatic and transparent to his program.

Instruction mnemonics and address mode symbols are sufficient for writing machine language programs. The programmer need not be concerned about conversion to binary digits; this is accomplished automatically by the PDP-11/70 assembler.

3.1 SINGLE OPERAND ADDRESSING

The instruction format for all single operand instructions such as clear, increment, test) is:



Bits 15 through 6 specify the operation code that defines the type of instruction to be executed.

Bits 5 through 0 form a six-bit field called the destination address field. This consists of two subfields:

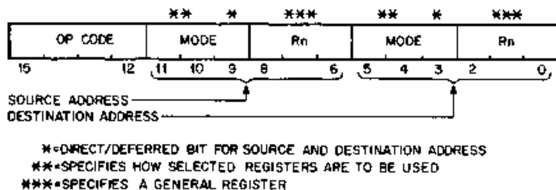
a) Bits 0 through 2 specify which of the eight general purpose registers is to be referenced by this instruction word.

b) Bits 4 and 5 specify how the selected register will be used (address mode). Bit 3 is set to indicate deferred (indirect) addressing.

3.2 DOUBLE OPERAND ADDRESSING

Operations which imply two operands (such as add, subtract, move and compare) are handled by instructions that specify two addresses. The

first operand is called the source operand, the second the destination operand. Bit assignments in the source and destination address fields may specify different modes and different registers. The Instruction format for the double operand instruction is:



The source address field is used to select the source operand, the first operand. The destination is used similarly, and locates the second operand and the result. For example, the instruction ADD A,B adds the contents (source operand) of location A to the contents (destination operand) of location B. After execution B will contain the result of the addition and the contents of A will be unchanged.

Examples in this section and further in this chapter use the following sample PDP-11 instructions:

Mnemonic	Description	Octal Code
CLR	clear (zero the specified destination)	0050nn
CLRB	clear byte (zero the byte in the specified destination)	1050nn
INC	increment (add 1 to contents of destination)	0052nn
INCB	increment byte (add 1 to the contents of destination byte)	1052nn
COM	complement (replace the contents of the destination by their logical complement; each 0 bit is set and each 1 bit is cleared)	0051nn
COMB	complement byte (replace the contents of the destination byte by their logical complement; each 0 bit is set and each 1 bit is cleared).	1051nn
ADD	add (add source operand to destination operand and store the result at destination address)	06mmnn

3.3 DIRECT ADDRESSING

The following table summarizes the four basic modes used with direct addressing.

DIRECT MODES

Mode	Name	Assembler Syntax	Function
0	Register	Rn	Register contains operand
2	Autoincrement	(Rn)+	Register is used as a pointer to sequential data then incremented.
4	Autodecrement	—(Rn)	Register is decremented and then used as a pointer.
6	Index	X(Rn)	Value X is added to (Rn) to produce address of operand. Neither X nor (Rn) are modified.

3.3.1 Register Mode

OPR Rn

With register mode any of the general registers may be used as simple accumulators and the operand is contained in the selected register. Since they are hardware registers, within the processor, the general registers operate at high speeds and provide speed advantages when used for operating on frequently-accessed variables. The PDP-11 assembler interprets and assembles instructions of the form OPR Rn as register mode operations. Rn represents a general register name or number and OPR is used to represent a general instruction mnemonic. Assembler syntax requires that a general register be defined as follows:

R0 = %0 (% sign indicates register definition)

R1 = %1

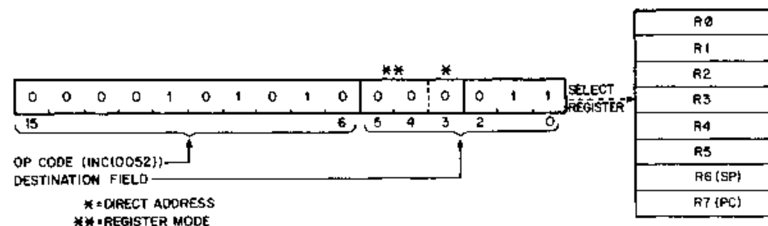
R2 = %2, etc.

Registers are typically referred to by name as R0, R1, R2, R3, R4, R5, R6 and R7. However R6 and R7 are also referred to as SP and PC, respectively.

Register Mode Examples
(all numbers in octal)

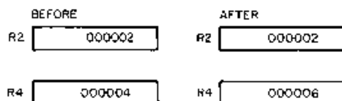
	Symbolic	Octal Code	Instruction Name
1.	INC R3	005203	Increment

Operation: Add one to the contents of general register 3



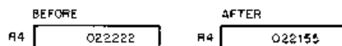
2. ADD R2,R4 060204 Add

Operation: Add the contents of R2 to the contents of R4.



3. COMB R4 105104 Complement Byte

Operation: One's complement bits 0-7 (byte) in R4. (When general registers are used, byte instructions only operate on bits 0-7; i.e. byte 0 of the register)



3.3.2 Autoincrement Mode

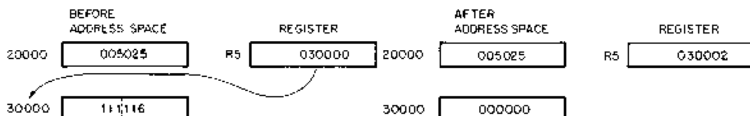
OPR (Rn)+

This mode provides for automatic stepping of a pointer through sequential elements of a table of operands. It assumes the contents of the selected general register to be the address of the operand. Contents of registers are stepped (by one for bytes, by two for words, always by two for R6 and R7) to address the next sequential location. The autoincrement mode is especially useful for array processing and stacks. It will access an element of a table and then step the pointer to address the next operand in the table. Although most useful for table handling, this mode is completely general and may be used for a variety of purposes.

Autoincrement Mode Examples

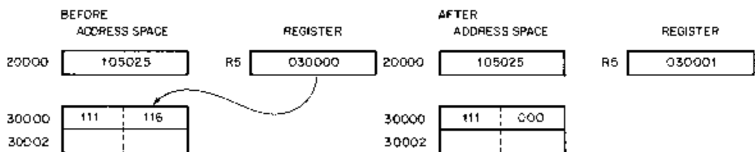
	Symbolic	Octal Code	Instruction Name
1.	CLR (R5)+	005025	Clear

Operation: Use contents of R5 as the address of the operand. Clear selected operand and then increment the contents of R5 by two.



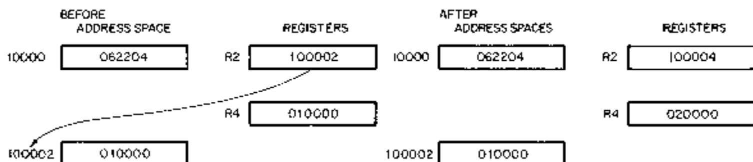
2. CLRB (R5)+ 105025 Clear Byte

Operation: Use contents of R5 as the address of the operand. Clear selected byte operand and then increment the contents of R5 by one.



3. ADD (R2)+, R4 062204 Add

Operation: The contents of R2 are used as the address of the operand which is added to the contents of R4. R2 is then incremented by two.



3.3.3 Autodecrement Mode

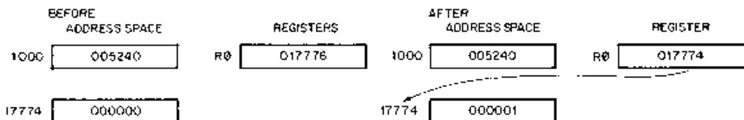
OPR—(Rn)

This mode is useful for processing data in a list in reverse direction. The contents of the selected general register are decremented (by two for word instructions, by one for byte instructions) and then used as the address of the operand. The choice of postincrement, predecrement features for the PDP-11 were not arbitrary decisions, but were intended to facilitate hardware/software stack operations.

Autodecrement Mode Examples

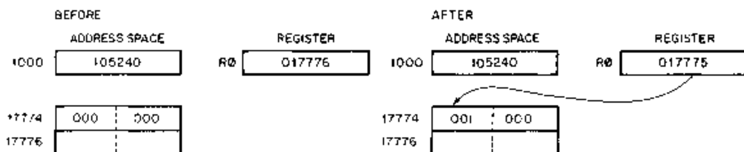
	Symbolic	Octal Code	Instruction Name
1.	INC—(R0)	005240	Increment

Operation: The contents of R0 are decremented by two and used as the address of the operand. The operand is increased by one.



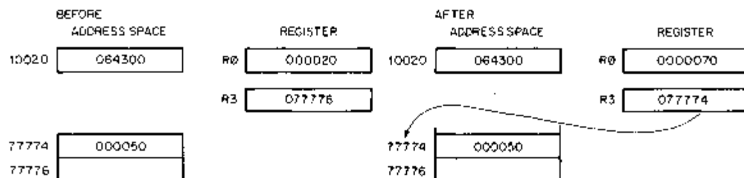
2. INCB—(R0) 105240 Increment Byte

Operation: The contents of R0 are decremented by one then used as the address of the operand. The operand byte is increased by one.



3. ADD—(R3),R0 064300 Add

Operation: The contents of R3 are decremented by 2 then used as a pointer to an operand (source) which is added to the contents of R0 (destination operand).



3.3.4 Index Mode

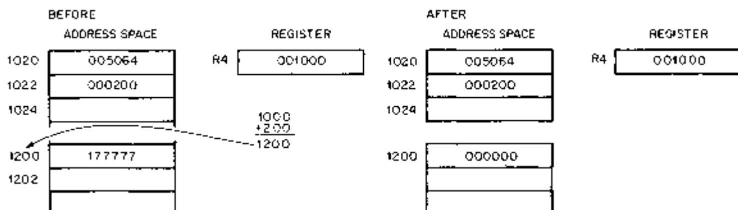
OPR X(Rn)

The contents of the selected general register, and an index word following the instruction word, are summed to form the address of the operand. The contents of the selected register may be used as a base for calculating a series of addresses, thus allowing random access to elements of data structures. The selected register can then be modified by program to access data in the table. Index addressing instructions are of the form OPR X(Rn) where X is the indexed word and is located in the memory location following the instruction word and Rn is the selected general register.

Index Mode Examples

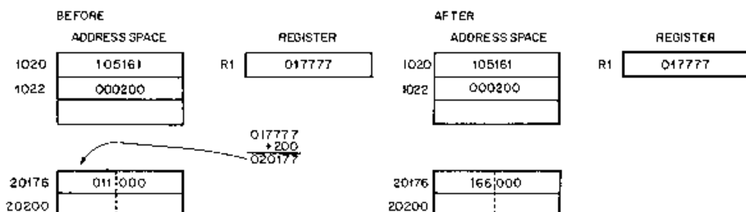
	Symbolic	Octal Code	Instruction Name
1.	CLR 200(R4)	005064 000200	Clear

Operation: The address of the operand is determined by adding 200 to the contents of R4. The location is then cleared.



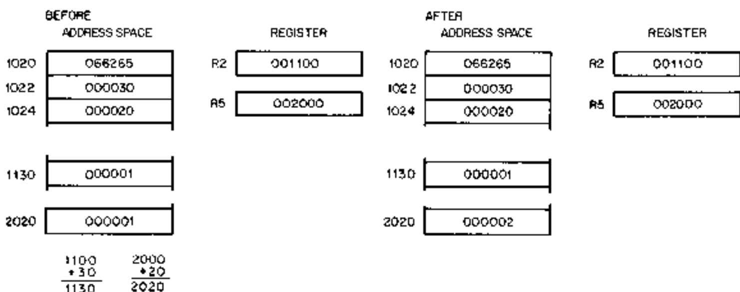
2. COMB 200(R1) 105161 Complement Byte
000200

Operation: The contents of a location which is determined by adding 200 to the contents of R1 are one's complemented (i.e. logically complemented).



3. ADD 30(R2), 20(R5)066265 Add
000030
000020

Operation: The contents of a location which is determined by adding 30 to the contents of R2 are added to the contents of a location which is determined by adding 20 to the contents of R5. The result is stored at the destination address, i.e. 20(R5)



3.4 DEFERRED (INDIRECT) ADDRESSING

The four basic modes may also be used with deferred addressing. Whereas in the register mode the operand is the contents of the selected register, in the register deferred mode the contents of the selected register is the address of the operand.

In the three other deferred modes, the contents of the register selects the address of the operand rather than the operand itself. These modes are therefore used when a table consists of addresses rather than operands. Assembler syntax for indicating deferred addressing is "@" or ( ). The following table summarizes the deferred versions of the basic modes:

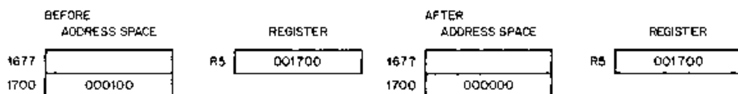
Mode	Name	Assembler Syntax	Function
1	Register Deferred	@Rn or (Rn)	Register contains the address of the operand
3	Autoincrement Deferred	@(Rn)+	Register is first used as a pointer to a word containing the address of the operand, then incremented (always by 2; even for byte instructions)
5	Autodecrement Deferred	@—(Rn)	Register is decremented (always by two; even for byte instructions) and then used as a pointer to a word containing the address of the operand
7	Index Deferred	@X(Rn)	Value X (stored in a word following the instruction) and (Rn) are added and the sum is used as a pointer to a word containing the address of the operand. Neither X nor (Rn) are modified.

Since each deferred mode is similar to its basic mode counterpart, separate descriptions of each deferred mode are not necessary. However, the following examples illustrate the deferred modes.

Register Deferred Mode Example

Symbolic	Octal Code	Instruction Name
CLR @R5	005015	Clear

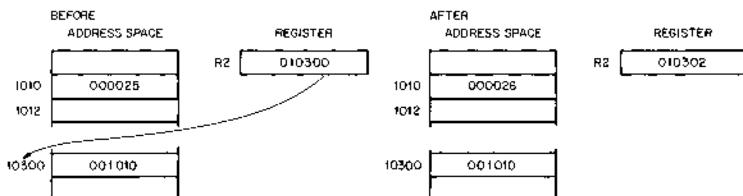
Operation: The contents of location specified in R5 are cleared.



Autoincrement Deferred Mode Example

Symbolic	Octal Code	Instruction Name
INC @(R2)+	005232	Increment

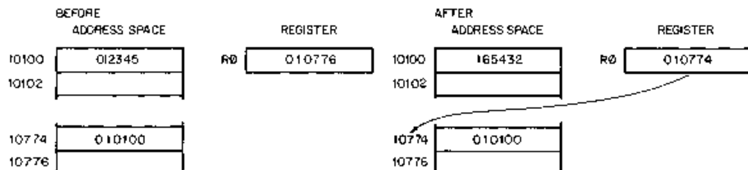
Operation: The contents of R2 are used as the address of the address of the operand. Operand is increased by one. Contents of R2 is incremented by 2.



Autodecrement Deferred Mode Example

Symbolic	Octal Code	Complement
COM @—(R0)	005150	

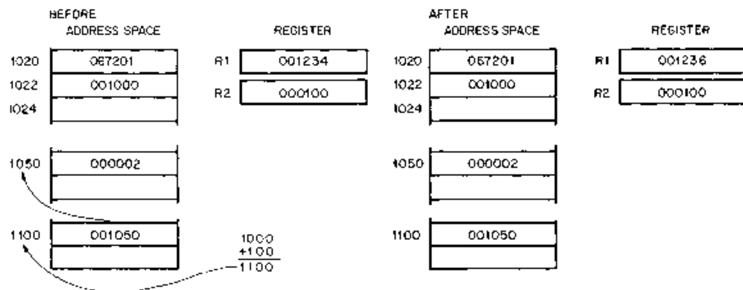
Operation: The contents of R0 are decremented by two and then used as the address of the address of the operand. Operand is one's complemented. (i.e. logically complemented)



Index Deferred Mode Example

Symbolic	Octal Code	Instruction Name
ADD @1000(R2),R1	067201 001000	Add

Operation: 1000 and contents of R2 are summed to produce the address of the address of the source operand the contents of which are added to contents of R1; the result is stored in R1.



3.5 USE OF THE PC AS A GENERAL REGISTER

Although Register 7 is a general purpose register, it doubles in function as the Program Counter for the PDP-11. Whenever the processor uses the program counter to acquire a word from memory, the program counter is automatically incremented by two to contain the address of the next word of the instruction being executed or the address of the next instruction to be executed. (When the program uses the PC to locate byte data, the PC is still incremented by two.)

The PC responds to all the standard PDP-11 addressing modes. However, there are four of these modes with which the PC can provide advantages for handling position independent code (PIC—see Chapter 9) and unstructured data. When regarding the PC these modes are termed immediate, absolute (or immediate deferred), relative and relative deferred, and are summarized below:

Mode	Name	Assembler Syntax	Function
2	Immediate	# n	Operand follows instruction.
3	Absolute	@ # A	Absolute Address follows instruction.
6	Relative	A	Address of A, relative to the instruction, follows the instruction.
7	Relative Deferred	@A	Address of location containing address of A, relative to the instruction follows the instruction.

The reader should remember that the special effect modes are the same as modes described in 3.3 and 3.4, but the general register selected is R7, the program counter.

When a standard program is available for different users, it often is helpful to be able to load it into different areas of core and run it there. PDP-11's can accomplish the relocation of a program very efficiently through the use of position independent code (PIC) which is written by using the PC addressing modes. If an instruction and its objects are moved in such a way that the relative distance between them is not altered, the same offset relative to the PC can be used in all positions in memory. Thus, PIC usually references locations relative to the current location.

The PC also greatly facilitates the handling of unstructured data. This is particularly true of the immediate and relative modes.

3.5.1 Immediate Mode

OPR #n,DD

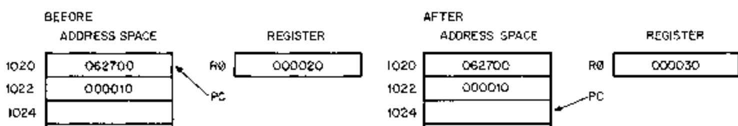
Immediate mode is equivalent to using the autoincrement mode with the PC. It provides time improvements for accessing constant operands by

including the constant in the memory location immediately following the instruction word.

Immediate Mode Example

	Symbolic	Octal Code	Instruction Name
	ADD # 10,R0	062700 000010	Add

Operation: The value 10 is located in the second word of the instruction and is added to the contents of R0. 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.



3.5.2 Absolute Addressing

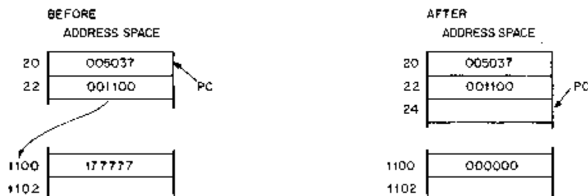
OPR @ # A

This mode is the equivalent of immediate deferred or autoincrement deferred using the PC. The contents of the location following the instruction are taken as the address of the operand. Immediate data is interpreted as an absolute address (i.e., an address that remains constant no matter where in memory the assembled instruction is executed).

Absolute Mode Examples

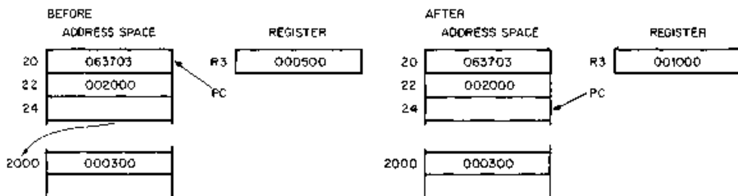
	Symbolic	Octal Code	Instruction Name
1.	CLR @#1100	005037 001100	Clear

Operation: Clear the contents of location 1100.



2. ADD @ # 2000, R3 063703
002000

Operation: Add contents of location 2000 to R3.



3.5.3 Relative Addressing

OPR A or

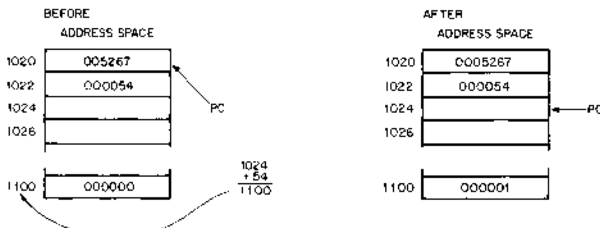
OPR X(PC) , where X is the location of A relative to the instruction.

This mode is assembled as index mode using R7. 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, but the number which, when added to the (PC), becomes the address of the operand. This mode is useful for writing position independent code (see Chapter 5) since the location referenced is always fixed relative to the PC. When instructions are to be relocated, the operand is moved by the same amount.

Relative Addressing Example

Symbolic	Octal Code	Instruction Name
INC A	005267 000054	Increment

Operation: To increment location A, contents of memory location immediately following instruction word are added to (PC) to produce address A. Contents of A are increased by one.



3.5.4 Relative Deferred Addressing

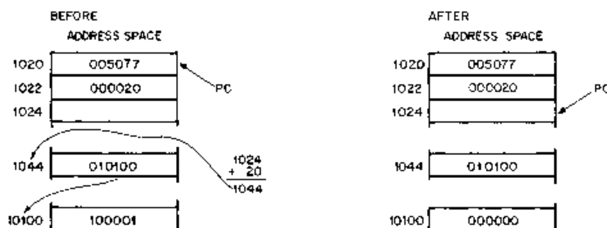
OPR@ or

OPR@X(PC), where x is location containing address of A, relative to the instruction.

This mode is similar to the relative mode, except that the second word of the instruction, when added to the PC, contains the address of the address of the operand, rather than the address of the operand.

Relative Deferred Mode Example

	Symbolic	Octal Code	Instruction Name
	CLR @A	005077 000020	Clear
Operation:		Add second word of instruction to PC to produce address of address of operand. Clear operand.	



3.6 USE OF STACK POINTER AS GENERAL REGISTER

The processor stack pointer (SP, Register 6) is in most cases the general register used for the stack operations related to program nesting. Autodecrement with Register 6 "pushes" data on to the stack and autoincrement with Register 6 "pops" data off the stack. Index mode with the SP permits random access of items on the stack. Since the SP is used by the processor for interrupt handling, it has a special attribute: autoincrements and autodecrements are always done in steps of two. Byte operations using the SP in this way simply leave odd addresses unmodified.

On the PDP-11/70 there are three R6 registers selected by the PS; but at any given time there is only one in operation.

The following table is a concise summary of the various PDP-11 addressing modes

DIRECT MODES

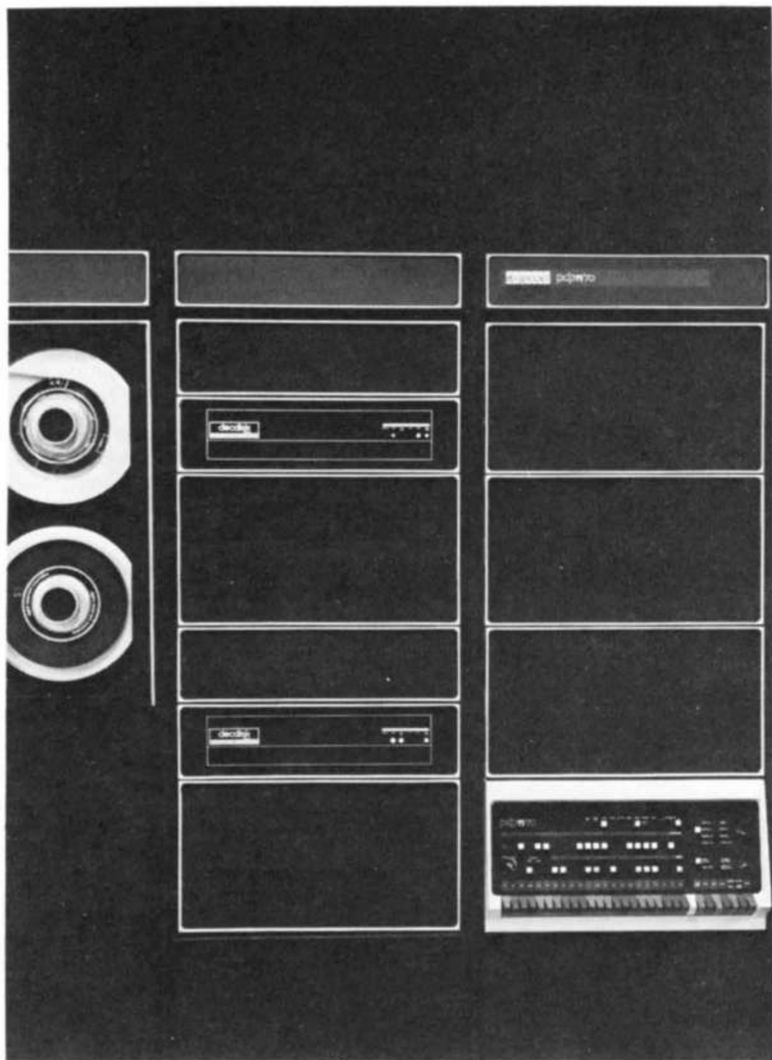
Mode	Name	Assembler Syntax	Function
0	Register	Rn	Register contains operand.
2	Autoincrement	(Rn) +	Register contains address of operand. Register contents incremented after reference.
4	Autodecrement	—(Rn)	Register contents decremented before reference register contains address of operand.
6	Index	X(Rn)	Value X (stored in a word following the instruction) is added to (Rn) to produce address of operand. Neither X nor (Rn) are modified.

DEFERRED MODES

Mode	Name	Assembler Syntax	Function
1	Register Deferred	@Rn or (Rn)	Register contains the address of the operand
3	Autoincrement Deferred	@(Rn) +	Register is first used as a pointer to A word containing the address of the operand, then incremented (always by 2; even for byte instructions)
5	Autodecrement Deferred	@—(Rn)	Register is decremented (always by two; even for byte instructions) and then used as a pointer to a word containing the address of the operand
7	Index Deferred	@X(Rn)	Value X (stored in a word following the instruction) and (Rn) are added and the sum is used as a pointer to a word containing the address of the operand. Neither X nor (Rn) are modified

PC ADDRESSING

Mode	Name	Assembler Syntax	Function
2	Immediate	#n	Operand follows instruction
3	Absolute	@#A	Absolute address follows instruction
6	Relative	A	Address of A, relative to the instruction, follows the instruction.
7	Relative Deferred	@A	Address of location containing address of A, relative to the instruction follows the instruction.



CHAPTER 4

INSTRUCTION SET

4.1 INTRODUCTION

This chapter describes the PDP-11/70 instructions in the following order:

Single Operand (4.4)
General, Shifts, Multiple Precision, Rotates

Double Operand (4.5)
Arithmetic Instructions, General Register Destination, Logical Instructions

Program Control Instructions (4.6)
Branches, Subroutines, Traps

Miscellaneous (4.7)

Condition Code Operators (4.8)

The specification for each instruction includes the mnemonic, octal code, binary code, a diagram showing the format of the instruction, a symbolic notation describing its execution and the effect on the condition codes, timing information, a description, special comments, and examples.

MNEMONIC: This is indicated at the top corner of each page. When the word instruction has a byte equivalent, the byte mnemonic is also shown.

INSTRUCTION FORMAT: A diagram accompanying each instruction shows the octal op code, the binary op code, and bit assignments. (Note that in byte instructions the most significant bit (bit 15) is always a 1.)

OPERATION: The operation of each instruction is described with a single notation. The following symbols are used:

( ) = contents of

src = source address

dst = destination address

loc = location

← = becomes

↑ = "is popped from stack"

↓ = "is pushed onto stack"

∧ = boolean AND

v = boolean OR

$\forall$ = exclusive OR

~ = boolean not

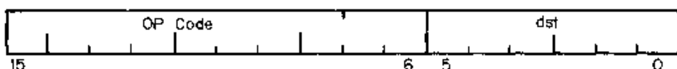
Reg or R = register

B = Byte

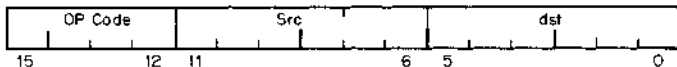
4.2 INSTRUCTION FORMATS

The major instruction formats are:

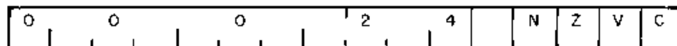
Single Operand Group



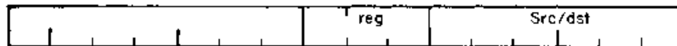
Double Operand Group



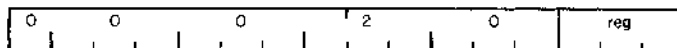
Condition Code Operators



Register-Source or Destination



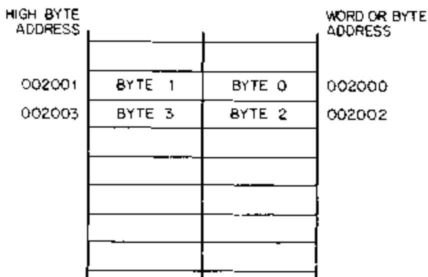
Subroutine Return



Branch

4.3 BYTE INSTRUCTIONS

The PDP-11 processor includes a full complement of instructions that manipulate byte operands. Since all PDP-11 addressing is byte-oriented, byte manipulation addressing is straightforward. Byte instructions with autoincrement or autodecrement direct addressing cause the specified register to be modified by one to point to the next byte of data. Byte operations in register mode access the low-order byte of the specified register. These provisions enable the PDP-11 to perform as either a word or byte processor. The numbering scheme for word and byte addresses in core memory is:



The most significant bit (Bit 15) of the instruction word is set to indicate a byte instruction.

Example:

Symbolic	Octal	
CLR	0050DD	clear word
CLRB	1050DD	clear byte

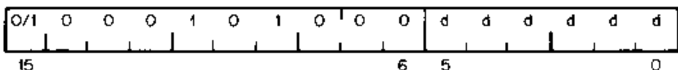
4.4 SINGLE OPERAND INSTRUCTIONS

4.4.1 Single Operand Arithmetic Instructions

General:	CLR CLRB	DEC DECB	INC INCB	NEG NEGB	TST TSTB	COM COMB
Shifts:	ASR ASRB	ASL ASLB	ASH	ASHC		
Multiple Precision:	ADC ADCB	SBC SBCB	SXT			
Rotates:	ROL ROLB	ROR RORB	SWAB			

CLR
CLRB

Clear destination n050DD



Operation: (dst) ← 0

Condition Codes:
N: cleared
Z: set
V: cleared
C: cleared

Description: Word: Contents of specified destination are replaced with zeroes.

Byte: Same

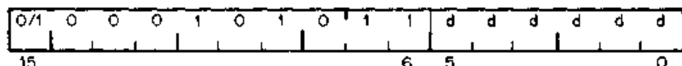
Example:

CLR R1

Before	After
(R1) = 177777	(R1) = 000000
N Z V C	N Z V C
1 1 1 1	0 1 0 0

DEC
DECB

Decrement destination n053DD



Operation: (dst) ←(dst) − 1

Condition Codes:
N: set if result is <0; cleared otherwise
Z: set if result is 0; cleared otherwise
V: set if (dst) was 100000; cleared otherwise
C: not affected

Description: Word: Subtract 1 from the contents of the destination
Byte: Same

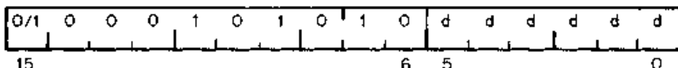
Example:

DEC R5

Before	After
(R5) = 000001	(R5) = 000000
N Z V C	N Z V C
1 0 0 0	0 1 0 0

INC
INCB

Increment destination n052DD

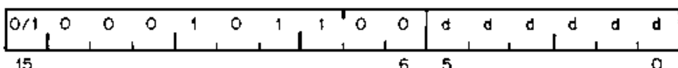


Operation:	(dst) ←(dst) + 1
Condition Codes:	N: set if result is <0; cleared otherwise Z: set if result is 0; cleared otherwise V: set if (dst) held 077777; cleared otherwise C: not affected
Description:	Word: Add one to contents of destination Byte: Same
Example:	INC R2

Before	After
(R2) = 000333	(R2) = 000334
N Z V C	N Z V C
0 0 0 0	0 0 0 0

NEGB
NEG

Negate destination n0054DD



Operation: (dst) ← −(dst)

Condition Codes:
N: set if the result is <0; cleared otherwise
Z: set if result is 0; cleared otherwise
V: set if the result is 100000; cleared otherwise
C: cleared if the result is 0; set otherwise

Description: Word: Replaces the contents of the destination address by its two's complement. Note that 100000 is replaced by itself (in two's complement notation the most negative number has no positive counterpart).
Byte: Same

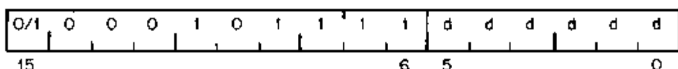
Example:

NEG R0

Before	After
(R0) = 000010	(R0) = 177770
N Z V C	N Z V C
0 0 0 0	1 0 0 1

TST
TSTB

Test destination n057DD



Operation: (dst) ←(dst)

Condition Codes:
N: set if the result is <0; cleared otherwise
Z: set if result is 0; cleared otherwise
V: cleared
C: cleared

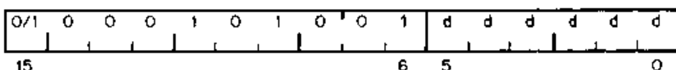
Description: Word: Sets the condition codes N and Z according to the contents of the destination address
Byte: Same

Example: TST R1

Before	After
(R1) = 012340	(R1) = 012340
N Z V C	N Z V C
0 0 1 1	0 0 0 0

COM
COMB

Complement destination n051DD



Operation: (dst) ← ~(dst)

Condition Codes: N: set if most significant bit of result is set; cleared otherwise
Z: set if result is 0; cleared otherwise
V: cleared
C: set

Description: Replaces the contents of the destination address by their logical complement (each bit equal to 0 is set and each bit equal to 1 is cleared)
Byte: Same

Example:

COM R0

Before	After
(R0) = 013333	(R0) = 164444
N Z V C	N Z V C
0 1 1 0	1 0 0 1

4.4.2 Shifts

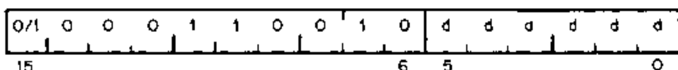
Scaling data by factors of two is accomplished by the shift instructions:

ASR—Arithmetic shift right	ASC—Multiple shift one word
ASL—Arithmetic shift left	ASC—Multiple shift one word

The sign bit (bit 15) of the operand is replicated in shifts to the right. The low order bit is filled with 0 in shifts to the left. Bits shifted out of the C bit, as shown in the following examples, are lost.

ASR
ASRB

Arithmetic Shift Right destination n062DD

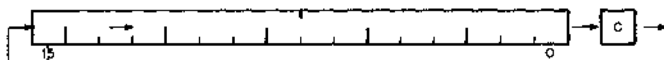


Operation: (dst) ←(dst) shifted one place to the right

Condition Codes: N: set if the high-order bit of the result is set (result < 0); cleared otherwise
Z: set if the result $= 0$; cleared otherwise
V: loaded from the Exclusive OR of the N-bit and C-bit (as set by the completion of the shift operation)
C: loaded from low-order bit of the destination

Description: Word: Shifts all bits of the destination right one place. Bit 15 is replicated. The C-bit is loaded from bit 0 of the destination. ASR performs signed division of the destination by two.

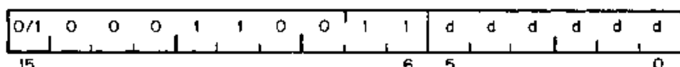
Word:



Byte:

ASL
ASLB

Arithmetic Shift Left destination n063DD

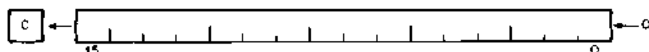


Operation: (dst)←(dst) shifted one place to the left

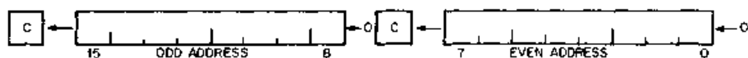
Condition Codes: N: set if high-order bit of the result is set (result < 0); cleared otherwise
Z: set if the result = 0; cleared otherwise
V: loaded with the exclusive OR of the N-bit and C-bit (as set by the completion of the shift operation)
C: loaded with the high-order bit of the destination

Description: Word: Shifts all bits of the destination left one place. Bit 0 is loaded with an 0. The C-bit of the status word is loaded from the most significant bit of the destination. ASL performs a signed multiplication of the destination by 2 with overflow indication.

Word:

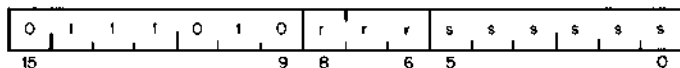


Byte:



ASH

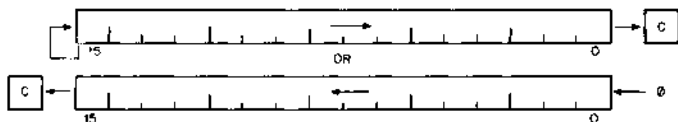
Shift Arithmetically 072RSS



Operation: R← R Shifted arithmetically NN places to right or left
Where NN = (src)

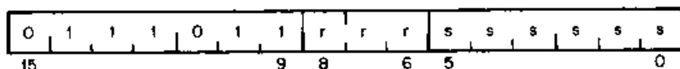
Condition Codes: N: set if result <0; cleared otherwise
Z: set if result = 0; cleared otherwise
V: set if sign of register changed during shift; cleared otherwise
C: loaded from last bit shifted out of register

Description: The contents of the register are shifted right or left the number of times specified by the source operand. The shift count is taken as the low order 6 bits of the source operand. This number ranges from −32 to +31. Negative is a right shift and positive is a left shift.



ASHC

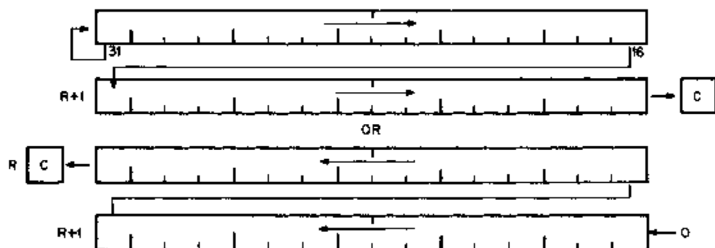
Arithmetic Shift Combined 073RSS



Operation: R, Rv1 ←R, Rv1 The double word is shifted NN places to the right or left, where NN = (src)

Condition Codes:
N: set if result <0; cleared otherwise
Z: set if result =0; cleared otherwise
V: set if sign bit changes during the shift; cleared otherwise
C: loaded with high order bit when left shift; loaded with low order bit when right shift (loaded with the last bit shifted out of the 32-bit operand)

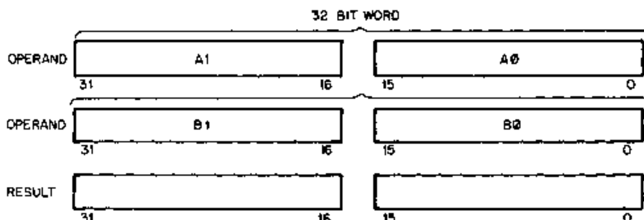
Description: The contents of the register and the register ORed with one are treated as one 32 bit word, R + 1 (bits 0-15) and R (bits 16-31) are shifted right or left the number of times specified by the shift count. The shift count is taken as the low order 6 bits of the source operand. This number ranges from −32 to +31. Negative is a right shift and positive is a left shift.
When the register chosen is an odd number the register and the register OR'ed with one are the same. In this case the right shift becomes a rotate. The 16 bit word is rotated right the number of bits specified by the shift count.



4.4.3 Multiple Precision

It is sometimes necessary to do arithmetic on operands considered as multiple words or bytes. The PDP-11 makes special provision for such operations with the instructions ADC (Add Carry) and SBC (Subtract Carry) and their byte equivalents.

For example two 16-bit words may be combined into a 32-bit double precision word and added or subtracted as shown below:



Example:

The addition of —1 and —1 could be performed as follows:

—1 = 37777777777

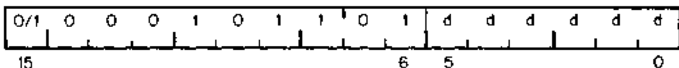
(R1) = 177777 (R2) = 177777 (R3) = 177777 (R4) = 177777

```
ADD    R1, R2
ADC    R3
ADD    R4,R3
```

1. After (R1) and (R2) are added, 1 is loaded into the C bit
2. ADC instruction adds C bit to (R3); (R3) = 0
3. (R3) and (R4) are added
4. Result is 37777777776 or —2

ADC
ADCB

Add Carry destination n055DD



Operation: (dst)←(dst) + (C)

Condition Codes:
N: set if result <0; cleared otherwise
Z: set if result =0; cleared otherwise
V: set if (dst) was 077777 and (C) was 1; cleared otherwise
C: set if (dst) was 177777 and (C) was 1; cleared otherwise

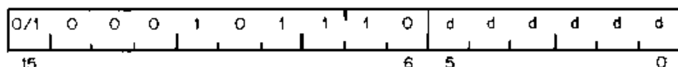
Description: Adds the contents of the C-bit into the destination. This permits the carry from the addition of the low-order words to be carried into the high-order result.
Byte: Same

Example: Double precision addition may be done with the following instruction sequence:

```
ADD    A0,B0      ; add low-order parts
ADC    B1         ; add carry into high-order
ADD    A1,B1      ; add high order parts
```

SBC
SBCB

Subtract Carry destination n056DD



Operation: (dst) ← (dst) — (C)

Condition Codes:
N: set if result <0; cleared otherwise
Z: set if result 0; cleared otherwise
V: set if (dst) was 100000; cleared otherwise
C: set if (dst) was 0 and C was 1; cleared otherwise

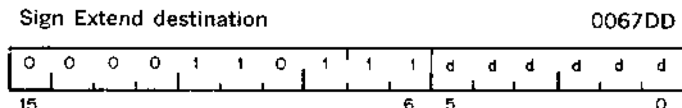
Description: Word: Subtracts the contents of the C-bit from the destination. This permits the carry from the subtraction of two low-order words to be subtracted from the high order part of the result.
Byte: Same

Example: Double precision subtraction is done by:

```
SUB    A0,B0
SBC    B1
SUB    A1,B1
```

SXT

Sign Extend destination 0067DD



Operation: (dst) ← 0 if N bit is clear
(dst) ← —1 N bit is set

Condition Codes: N: unaffected
Z: set if N bit clear
V: cleared
C: unaffected

Description: If the condition code bit N is set then a —1 is placed in the destination operand; if N bit is clear, then a 0 is placed in the destination operand. This instruction is particularly useful in multiple precision arithmetic because it permits the sign to be extended through multiple words.

Example: SXT A

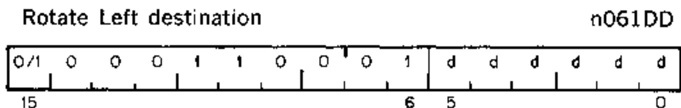
Before	After
(A) = 012345	(A) = 177777
N Z V C	N Z V C
1 0 0 0	1 0 0 0

4.4.4 Rotates

The rotate instructions operate on the destination word and the C bit as though they formed a 17-bit "circular buffer." These instructions facilitate sequential bit testing and detailed bit manipulation.

ROL
ROLB

Rotate Left destination n061DD



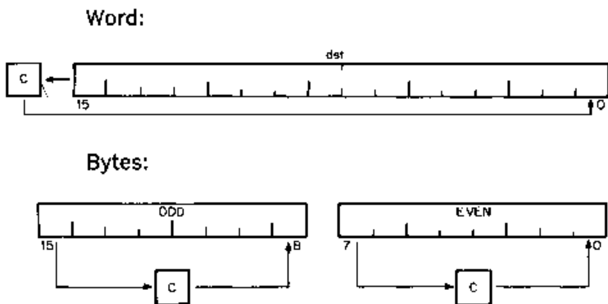
Condition Codes:

N: set if the high-order bit of the result word is set (result <0): cleared otherwise
Z: set if all bits of the result word = 0; cleared otherwise
V: loaded with the Exclusive OR of the N-bit and C-bit (as set by the completion of the rotate operation)
C: loaded with the high-order bit of the destination

Description:

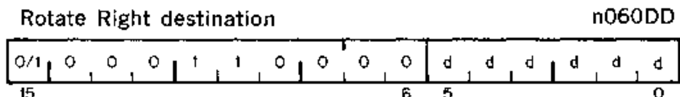
Word: Rotate all bits of the destination left one place. Bit 15 is loaded into the C-bit of the status word and the previous contents of the C-bit are loaded into Bit 0 of the destination.
Byte: Same

Example:



ROR
RORB

Rotate Right destination n060DD

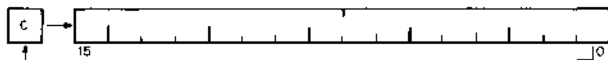


Condition Codes: N: set if the high-order bit of the result is set (result < 0); cleared otherwise
Z: set if all bits of result = 0; cleared otherwise
V: loaded with the Exclusive OR of the N-bit and C-bit (as set by the completion of the rotate operation)
C: loaded with the low-order bit of the destination

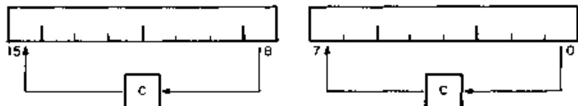
Description: Rotates all bits of the destination right one place. Bit 0 is loaded into the C-bit and the previous contents of the C-bit are loaded into bit 15 of the destination.
Byte: Same

Example:

Word:

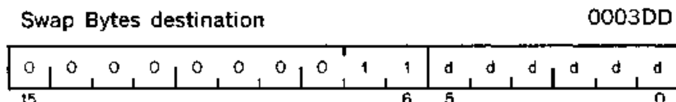


Byte:



SWAB

Swap Bytes destination 0003DD



Operation: Byte 1/Byte 0 ←Byte 0/Byte 1

Condition Codes: N: set if high-order bit of low-order byte (bit 7) of result is set; cleared otherwise
Z: set if low-order byte of result = 0; cleared otherwise
V: cleared
C: cleared

Description: Exchanges high-order byte and low-order byte of the destination word (destination must be a word address).

Example:

SWAB R1

Before	After
(R1) = 077777	(R1) = 177577
N Z V C	N Z V C
1 1 1 1	0 0 0 0

4.5 DOUBLE OPERAND INSTRUCTIONS

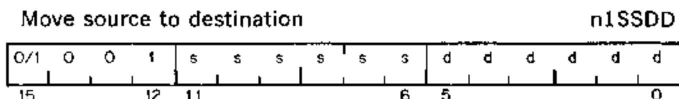
Double operand instructions provide an instruction (and time) saving facility since they eliminate the need for "load" and "save" sequences such as those used in accumulator-oriented machines.

General:	MOV	ADD	SUB	CMP
	MOVB			CMPB
Register Destination:	MUL	DIV	XOR	
Logical:	BIS	BIT	BIC	
	BISB	BITB	BICB	

4.5.1 Double Operand General Instructions

MOV
MOVB

Move source to destination n1SSDD



Operation: (dst)←(src)

Condition Codes:
N: set if (src) <0; cleared otherwise
Z: set if (src) = 0; cleared otherwise
V: cleared
C: not affected

Description: Word: Moves the source operand to the destination location. The previous contents of the destination are lost. The contents of the source address are not affected.
Byte: Same as MOV. The MOVB to a register (unique among byte instructions) extends the most significant bit of the low order byte (sign extension). Otherwise MOVB operates on bytes exactly as MOV operates on words.

Example:

MOV XXX,R1 ; loads Register 1 with the contents of memory location; XXX represents a programmer-defined mnemonic used to represent a memory location

MOV #20,R0 ; loads the number 20 into Register 0; "#" indicates that the value 20 is the operand

MOV @#20,—(R6) ; pushes the operand contained in location 20 onto the stack

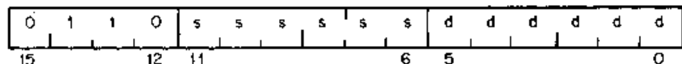
MOV (R6)+,@#177566 ; pops the operand off a stack and moves it into memory location 177566 (terminal print buffer)

MOV R1,R3 ; performs an interregister transfer

MOVB @#177562,@#177566 ; moves a character from terminal keyboard buffer to terminal buffer

ADD

Add source to destination 06SSDD



Operation: (dst) ←(src) + (dst)

Condition Codes:
N: set if result <0; cleared otherwise
Z: set if result = 0; cleared otherwise
V: set if there was arithmetic overflow as a result of the operation; that is both operands were of the same sign and the result was of the opposite sign; cleared otherwise
C: set if there was a carry from the most significant bit of the result; cleared otherwise

Description: Adds the source operand to the destination operand and stores the result at the destination address. The original contents of the destination are lost. The contents of the source are not affected. Two's complement addition is performed.

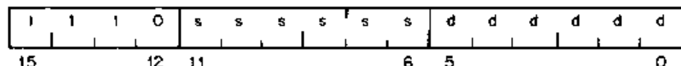
Examples:

Add to register:	ADD	20,R0
Add to memory:	ADD	R1,XXX
Add register to register:	ADD	R1,R2
Add memory to memory:	ADD	@#17750,XXX

XXX is a programmer-defined mnemonic for a memory location.

SUB

Subtract source from destination 16SSDD



Operation: (dst)←(dst)−(src)

Condition Codes:
N: set if result <0; cleared otherwise
Z: set if result =0; cleared otherwise
V: set if there was arithmetic overflow as a result of the operation, that is if operands were of opposite signs and the sign of the source was the same as the sign of the result; cleared otherwise
C: cleared if there was a carry from the most significant bit of the result; set otherwise

Description: Subtracts the source operand from the destination operand and leaves the result at the destination address. The original contents of the destination are lost. The contents of the source are not affected. In double-precision arithmetic the C-bit, when set, indicates a "borrow"

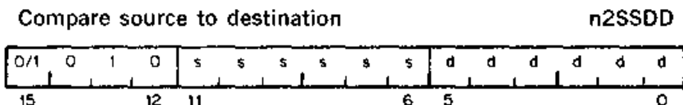
Example:

SUB R1, R2

Before	After
(R1) = 011111	(R1) = 011111
(R2) = 012345	(R2) = 001234
N Z V C	N Z V C
1 1 1 1	0 0 0 0

CMP
CMPB

Compare source to destination n2SSDD



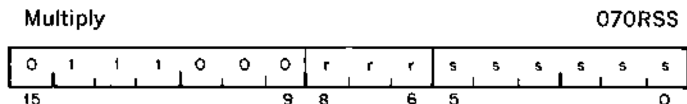
Operation: (src)−(dst) [in detail, (src) + ~ (dst) + 1]

Condition Codes:
N: set if result <0; cleared otherwise
Z: set if result =0; cleared otherwise
V: set if there was arithmetic overflow; that is, operands were of opposite signs and the sign of the destination was the same as the sign of the result; cleared otherwise
C: cleared if there was a carry from the most significant bit of the result; set otherwise

Description: Compares the source and destination operands and sets the condition codes, which may then be used for arithmetic and logical conditional branches. Both operands are unaffected. The only action is to set the condition codes. The compare is customarily followed by a conditional branch instruction.
Note that unlike the subtract instruction the order of operation is (src)—(dst), not (dst)—(src).

MUL

Multiply 070RSS



Operation: R, Rvl ← R x(src)

Conditon Codes:
N: set if product is <0; cleared otherwise
Z: set if product is 0; cleared otherwise
V: cleared
C: set if the result is less than -2^{15} or greater than or equal to $2^{15}-1$.

Description: The contents of the destination register and source taken as two's complement integers are multiplied and stored in the destination register and the succeeding register (if R is even). If R is odd only the low order product is stored. Assembler syntax is: MUL S,R.
(Note that the actual destination is R,Rv1 which reduces to just R when R is odd.)

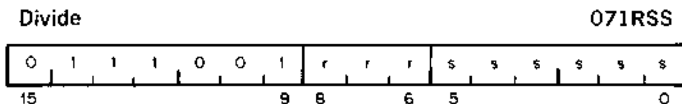
Example: 16-bit product (R is odd)

```
CLC              ;Clear carry condition code
MOV #400,R1
MUL #10,R1
BCS ERROR        ;Carry will be set if
                 ;product is less than
                 ;—2^15 or greater than or equal
                 to 2^15
                 ;no significance lost
```

Before	After
(R1) = 000400	(R1) = 004000

DIV

Divide 071RSS



Operation: R, Rv1 ←R , Rv1 / (src)

Condition Codes:
N: set if quotient <0; cleared otherwise
Z: set if quotient =0; cleared otherwise
V: set if source =0 or if the absolute value of the register is larger than the absolute value of the source. (In this case the instruction is aborted because the quotient would exceed 15 bits.)
C: set if divide 0 attempted; cleared otherwise

Description: The 32-bit two's complement integer in R and Rv1 is divided by the source operand. The quotient is left in R; the remainder in Rv1. Division will be performed so that the remainder is of the same sign as the dividend. R must be even.

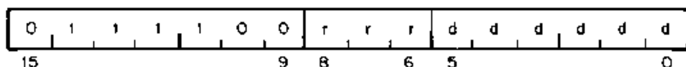
Example:

```
CLR R0
MOV #20001,R1
DIV #2,R0
```

Before	After	
(R0) = 000000	(R0) = 010000	Quotient
(R1) = 020001	(R1) = 000001	Remainder

XOR

Exclusive Or 074RDD



Operation: (dst) ←R∨(dst)

Condition Codes:
N: set if the result <0; cleared otherwise
Z: set if result =0; cleared otherwise
V: cleared
C: unaffected

Description: The exclusive OR of the register and destination operand is stored in the destination address. Contents of register are unaffected. Assembler format is: XOR R,D

Example:

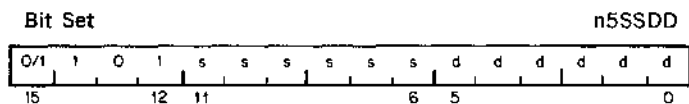
XOR R0,R2

Before	After
(R0) = 001234	(R0) = 001234
(R2) = 001111	(R2) = 000325

4.5.2 Logical Instructions
These instructions have the same format as the double operand arithmetic group. They permit operations on data at the bit level.

BIS
BISB

Bit Set n5SSDD



Operation: (dst) ←(src) v (dst)

Condition Codes: N: set if high-order bit of result set, cleared otherwise
Z: set if result = zero; cleared otherwise
V: cleared
C: not affected

Description: Performs "Inclusive OR" operation between the source and destination operands and leaves the result at the destination address; that is, corresponding bits set in the source are set in the destination. The content of the destination are lost.

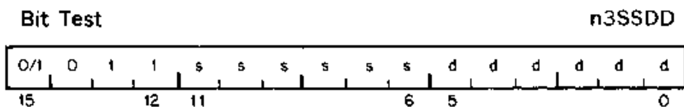
Example:

BIS R0,R1

Before	After
(R0) = 001234	(R0) = 001234
(R1) = 001111	(R1) = 001335
N Z V C	N Z V C
0 0 0 0	0 0 0 0

BIT
BITB

Bit Test n3SSDD



Operation: (dst)∧(src)

Condition Codes:
N: set if high-order bit of result set; cleared otherwise
Z: set if result =0; cleared otherwise
V: cleared
C: not affected

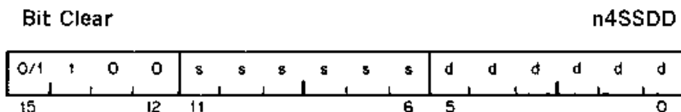
Description: Performs logical "and" comparison of the source and destination operands and modifies condition codes accordingly. Neither the source nor destination operands are affected. The BIT instruction may be used to test whether any of the corresponding bits that are set in the destination are also set in the source or whether all corresponding bits set in the destination are clear in the source.

Example:

```
BIT   #30,R3          ; test bits 3 and 4 of R3
                      ; to see if both are off
```

BIC
BICB

Bit Clear n4SSDD



Operation: (dst) ← ~(src) ∧ (dst)

Condition Codes: N: set if high order bit of result set; cleared otherwise
Z: set if result =0; cleared otherwise
V: cleared
C: not affected

Description: Clears each bit in the destination that corresponds to a set bit in the source. The original contents of the destination are lost. The contents of the source are unaffected.

Example: BIC R3,R4

Before	After
(R3) = 001234	(R3) = 001234
(R4) = 001111	(R4) = 000101
N Z V C	N Z V C
1 1 1 1	0 0 0 1

4.6 PROGRAM CONTROL INSTRUCTIONS

4.6.1 Branches

The instruction causes a branch to a location defined by the sum of the offset (multiplied by 2) and the current contents of the Program Counter if:

a) the branch instruction is unconditional

b) it is conditional and the conditions are met after testing the condition codes (status word).

The offset is the number of words from the current contents of the PC. Note that the current contents of the PC point to the word following the branch instruction.

Although the PC expresses a byte address, the offset is expressed in words. The offset is automatically multiplied by two to express bytes before it is added to the PC. Bit 7 is the sign of the offset. If it is set, the offset is negative and the branch is done in the backward direction. Similarly if it is not set, the offset is positive and the branch is done in the forward direction.

The 8-bit offset allows branching in the backward direction by 200_8 words (400_8 bytes) from the current PC, and in the forward direction by 177_8 words (376_8 bytes) from the current PC.

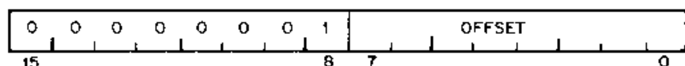
The PDP-11 assembler handles address arithmetic for the user and computes and assembles the proper offset field for branch instructions in the form:

Bxx   loc

Where "Bxx" is the branch instruction and "loc" is the address to which the branch is to be made. The assembler gives an error indication in the instruction if the permissible branch range is exceeded. Branch instructions have no effect on condition codes.

BR

Branch (unconditional) 0004 loc



Operation: PC ← PC + (2 x offset)

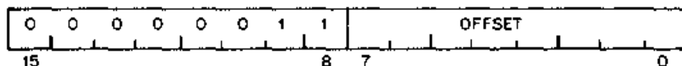
Description: Provides a way of transferring program control within a range of −128 to +127 words with a one word instruction.

Simple Conditional Branches

BEQ
BNE
BMI
BPL
BCS
BCC
BVS
BVC

BEQ

Branch on Equal (zero) 0014 offset



Operation: PC ← PC + (2 x offset) if Z = 1

Condition Codes: Unaffected

Description: Tests the state of the Z-bit and causes a branch if Z is set. As an example, it is used to test equality following a CMP operation, to test that no bits set in the destination were also set in the source following a BIT operation, and generally, to test that the result of the previous operation was zero.

Example:

```
CMP   A,B          ; compare A and B
BEQ   C            ; branch if they are equal
```

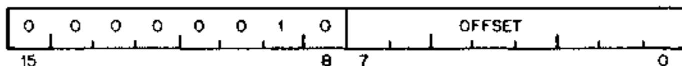
will branch to C if A = B (A − B = 0)
and the sequence

```
ADD   A,B          ; add A to B
BEQ   C            ; branch if the result = 0
```

will breach to C if A + B = 0.

BNE

Branch Not Equal (Zero) 0010 offset



Operation: PC ← PC + (2 x offset) if Z = 0

Condition Codes: Unaffected

Description: Tests the state of the Z-bit and causes a branch if the Z-bit is clear. BNE is the complementary operation to BEQ. It is used to test inequality following a CMP, to test that some bits set in the destination were also in the source, following a BIT, and generally, to test that the result of the previous operation was not zero.

Example:

```
CMP   A,B          ; compare A and B
BNE   C            ; branch if they are not equal
```

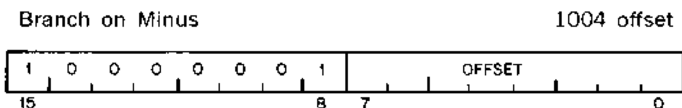
will branch to C if A ≠ B
and the sequence

```
ADD   A,B          ; add A to B
BNE   C            ; branch if the result is not equal
                     to 0
```

will branch to C if A + B ≠ 0

BMI

Branch on Minus 1004 offset



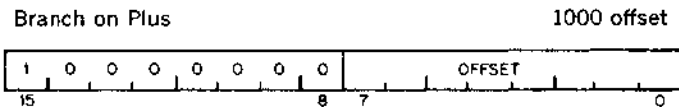
Operation: PC ← PC + (2 x offset) if N = 1

Condition Codes: Unaffected

Description: Tests the state of the N-bit and causes a branch if N is set. It is used to test the sign (most significant bit) of the result of the previous operation, branching if negative.

BPL

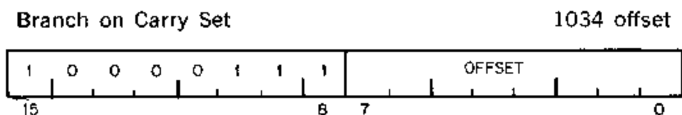
Branch on Plus 1000 offset



Operation: PC ← PC + (2 x offset) if N = 0

Description: Tests the state of the N-bit and causes a branch if N is clear. BPL is the complementary operation of BMI.

BCS

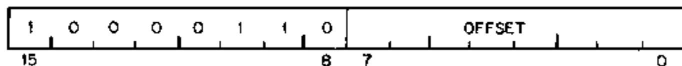


Operation: PC ← PC + (2 x offset) if C = 1

Description: Tests the state of the C-bit and causes a branch if C is set. It is used to test for a carry in the result of a previous operation.

BCC

Branch on Carry Clear 1030 offset

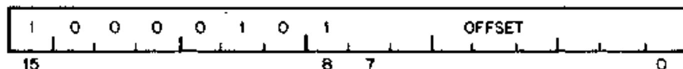


Operation: PC ← PC + (2 x offset) if C = 0

Description: Tests the state of the C-bit and causes a branch if C is clear. BCC is the complementary operation to BCS

BVS

Branch on Overflow Set 1024 offset

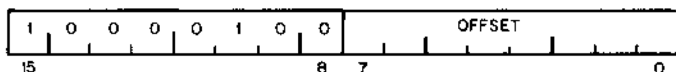


Operation: PC ← PC + (2 x offset) if V = 1

Description: Tests the state of V bit (overflow) and causes a branch if the V bit is set. BVS is used to detect arithmetic overflow in the previous operation.

BVC

Branch on Overflow Clear 1020 offset



Operation: PC ← PC + (2 x offset) if V = 0

Description: Tests the state of the V bit and causes a branch if the V bit is clear. BVC is complementary operation to BVS.

Signed Conditional Branches

Particular combinations of the condition code bits are tested with the signed conditional branches. These instructions are used to test the results of instructions in which the operands were considered as a signed (two's complement) values.

Note that the sense of signed comparisons differs from that of unsigned comparisons in that in signed 16-bit, two's complement arithmetic the sequence of values is as follows:

```
largest      077777
             077776
positive     .
             .
             .
             000001
             000000
             177777
             177776
             .
negative     .
             .
             100001
smallest     100000
```

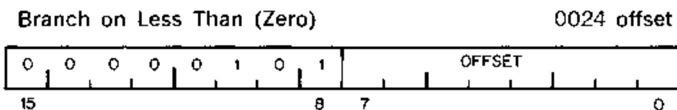
whereas in unsigned 16-bit arithmetic the sequence is considered to be

```
highest      177777
             .
             .
             .
             .
             .
             .
             000002
             000001
lowest       000000
```

The signed conditional branch instructions are:

BLT	BGE
BLE	BGT

Branch on Less Than (Zero) 0024 offset

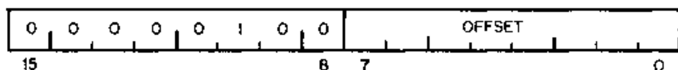


Operation: PC ← PC + (2 x offset) if N v V = 1

Description: Causes a branch if the "Exclusive Or" of the N and V bits are 1. Thus BLT will always branch following an operation that added two negative numbers, even if overflow occurred.
In particular, BLT will always cause a branch if it follows a CMP instruction operating on a negative source and a positive destination (even if overflow occurred). Further, BLT will never cause a branch when it follows a CMP instruction operating on a positive source and negative destination. BLT will not cause a branch if the result of the previous operation was zero (without overflow).

BGE

Branch on Greater than or Equal (zero) 0020 offset

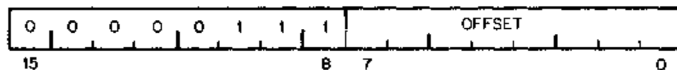


Operation: PC ← PC + (2 x offset) if N v V = 0

Description: Causes a branch if N and V are either both clear or both set. BGE is the complementary operation to BLT. Thus BGE will always cause a branch when it follows an operation that caused addition of two positive numbers. BGE will also cause a branch on a zero result.

BLE

Branch on Less than or Equal (zero) 0034 offset

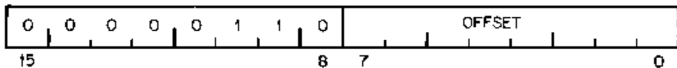


Operation: $PC \leftarrow PC + (2 \times offset)$ if $Z \vee (N \veebar V) = 1$

Description: Operation is similar to BLT but in addition will cause a branch if the result of the previous operation was zero.

BGT

Branch on Greater Than (zero) 0030 offset



Operation $PC \leftarrow PC + (2 \times offset)$ if $Z \vee (N \forall V) = 0$

Description: Operation of BGT is similar to BGE, except BGT will not cause a branch on a zero result.

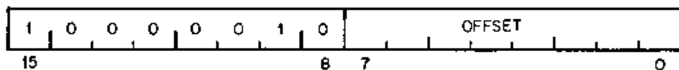
Unsigned Conditional Branches

The Unsigned Conditional Branches provide a means for testing the result of comparison operations in which the operands are considered as unsigned values.

BHI
BLOS
BHIS
BLO

BHI

Branch on Higher 1010 offset

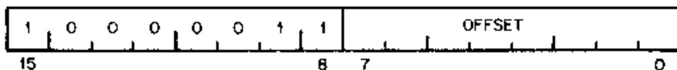


Operation: PC ← PC + (2 x offset) if C = 0 and Z = 0

Description: Causes a branch if the previous operation caused neither a carry nor a zero result. This will happen in comparison (CMP) operations as long as the source has a higher unsigned value than the destination.

BLOS

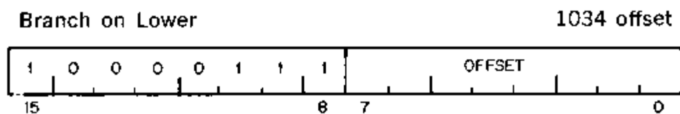
Branch on Lower or Same 1014 offset



Operation: PC ← PC + (2 x offset) if C v Z = 1

Description: Causes a branch if the previous operation caused either a carry or a zero result. BLOS is the complementary operation to BHI. The branch will occur in comparison operations as long as the source is equal to, or has a lower unsigned value than the destination.

BLO

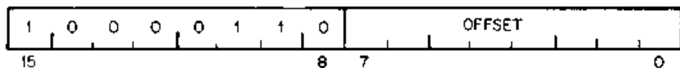


Operation: PC ← PC + (2 x offset) if C = 1

Description: BLO is same instruction as BCS. This mnemonic is included only for convenience.

BHIS

Branch on Higher or Same 1030 offset



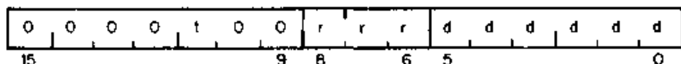
Operation: PC ← PC + (2 x offset) if C = 0

Description: BHIS is the same instruction as BCC. This mnemonic is included only for convenience.

4.6.2 Subroutine Instructions

The subroutine call in the PDP-11 provides for automatic nesting of subroutines, reentrancy, and multiple entry points. Subroutines may call other subroutines (or indeed themselves) to any level of nesting without making special provision for storage or return addresses at each level of subroutine call. The subroutine calling mechanism does not modify any fixed location in memory, thus providing for reentrancy. This allows one copy of a subroutine to be shared among several interrupting processes. For more detailed description of subroutine programming see Chapter 5.

Jump to Sub Routine 004 reg. dst



Operation:

(tmp)←(dst) (tmp is an internal processor register)

↓(SP)←reg (push reg contents onto processor stack)

reg←PC (PC holds location following JSR; this address now put in reg)

PC←(tmp) (PC now points to subroutine address)

Description:

In execution of the JSR, the old contents of the specified register (the "LINKAGE POINTER") are automatically pushed onto the processor stack and new linkage information placed in the register. Thus subroutines nested within subroutines to any depth may all be called with the same linkage register. There is no need either to plan the maximum depth at which any particular subroutine will be called or to include instructions in each routine to save and restore the linkage pointer. Further, since all linkages are saved in a reentrant manner on the processor stack, execution of a subroutine may be interrupted, the same subroutine reentered and executed by an interrupt service routine. Execution of the initial subroutine can then be resumed when other requests are satisfied. This process (called nesting) can proceed to any level.

In both JSR and JMP instructions the destination address is used to load the program counter, R7. Thus for example a JSR in destination mode 1 for general register R1 (where (R1) = 100), will access a subroutine at location 100. This is effectively one level less of deferral than operate instructions such as ADD.

A subroutine called with a JSR reg,dst instruction can access the arguments following the call with either autoincrement addressing, (reg) +, (if arguments are accessed sequentially) or by indexed

addressing, X(reg), (if accessed in random order). These addressing modes may also be deferred, @(reg) + and @X(reg) if the parameters are operand addresses rather than the operand themselves.

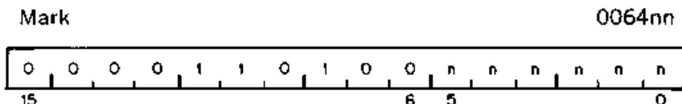
JSR PC, dst is a special case of the PDP-11 subroutine call suitable for subroutine calls that transmit parameters through the general registers. The SP and the PC are the only registers that may be modified by this call.

Another special case of the JSR instruction is JSR PC, @(SP) + which exchanges the top element of the processor stack and the contents of the program counter. Use of this instruction allows two routines to swap program control and resume operation when recalled where they left off. Such routines are called "co-routines."

Return from a subroutine is done by the RTS instruction. RTS reg loads the contents of reg into the PC and pops the top element of the processor stack into the specified register.

MARK

Mark 0064nn



Operation:

SP←PC + 2xnn nn = number of parameters
PC←R5
R5←(SP)↑

Condition Codes: unaffected

Description: Used as part of the standard PDP-11 subroutine return convention. MARK facilitates the stack clean up procedures involved in subroutine exist. Assembler format is: MARK N

Example:

```
MOV   R5,—(SP)         ;place old R5 on stack
MOV   P1,—(SP)         ;place N parameters
MOV   P2,—(SP)         ;on the stack to be
                       ;used there by the
                       ;subroutine
MOV   PN,—(SP)
MOV   =MARKN,—(SP)     ;places the instruction
                       ;MARK N on the stack
MOV   SP ,R5           ;set up address at Mark N
                        instruction
JSR   PC,SUB           ;jump to subroutine
```

At this point the stack is as follows:

OLD R5
P1
PN
MARK N
OLD PC

And the program is at the address SUB which is the beginning of the subroutine.

```
SUB:               ;execution of the subroutine it-
                    self

       RTS R5      ;the return begins: this causes
```

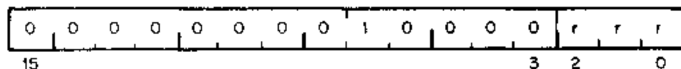
the contents of R5 to be placed in the PC which then results in the execution of the instruction MARK N. The contents of the old PC are placed in R5.

MARK N causes: (1) the stack pointer to be adjusted to point to the old R5 value; (2) the value now in R5 (the old PC) to be placed in the PC; and (3) contents of the old R5 to be popped into R5 thus completing the return from subroutine.

Note: If Memory Management is in use a stack must be in I and D spaces (Chapter 6) to execute the MARK instruction.

RTS

Return from Subroutine 00020 Reg



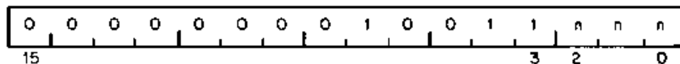
Operation: PC ← reg
reg ← (SP)↑

Description: Loads contents of reg into PC and pops the top element of the processor stack into the specified register.
Return from a non-reentrant subroutine is typically made through the same register that was used in its call. Thus, a subroutine called with a JSR PC, dst exits with a RTS PC and a subroutine called with a JSR R5, dst, may pick up parameters with addressing modes (R5)+, X(R5), or @X(R5) and finally exits, with an RTS R5.

4.6.3 Program Control Instructions

SPL
JMP
SOB

Set Priority Level 00023N



Operation: PS (bits 7-5)←Priority

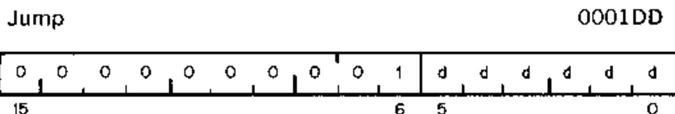
Condition Codes: not affected

Description The least significant three bits of the instruction are loaded into the Program Status Word (PS) bits 7-5 thus causing a changed priority. The old priority is lost.
Assembler syntax is: SPL N

Note: This instruction is a no op in User and Supervisor modes.

JMP

Jump 0001DD



Operation: PC←(dst)

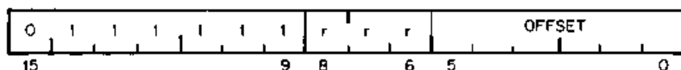
Condition Codes: not affected

Description: JMP provides more flexible program branching than provided with the branch instructions. Control may be transferred to any location in memory (no range limitation) and can be accomplished with the full flexibility of the addressing modes, with the exception of register mode 0. Execution of a jump with mode 0 will cause an "illegal" instruction" condition. (Program control cannot be transferred to a register.) Register deferred mode is legal and will cause program control to be transferred to the address held in the specified register. Note that instructions are word data and must therefore be fetched from an even-numbered address. A "boundary error" trap condition will result when the processor attempts to fetch an instruction from an odd address.

Deferred index mode JMP instructions permit transfer of control to the address contained in a selectable element of a table of dispatch vectors.

SOB

Subtract One and Branch 077R offset



Operation: R ← R −1 if this result ≠ 0 then PC ← PC −(2x offset)

Condition Codes: unaffected

Description: The register is decremented. If it is not equal to 0, twice the offset is subtracted from the PC (now pointing to the following word). The offset is interpreted as a six bit positive number. This instruction provides a fast, efficient method of loop control. Assembler syntax is:

SOB R,A

Where A is the address to which transfer is to be made if the decremented R is not equal to 0. Note that the SOB instruction can not be used to transfer control in the forward direction.

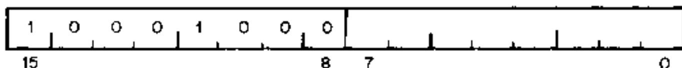
4.6.4 Traps

Trap instructions provide for calls to emulators, I/O monitors, debugging packages, and user-defined interpreters. A trap is effectively an interrupt generated by software. When a trap occurs the contents of the current Program Counter (PC) and Program Status Word (PS) are pushed onto the processor stack and replaced by the contents of a two-word trap vector containing a new PC and new PS. The return sequence from a trap involves executing an RTI or RTT instruction which restores the old PC and old PS by popping them from the stack. Trap vectors are located permanently assigned fixed address.

TRAP
EMT
BPT
IOT
RTI
RTT

EMT

Emulator Traps 104000-104377



Operation:

↓(SP)←PS
↓(SP)←PC
PC←(30)
PS←(32)

Condition Codes:

N: loaded from trap vector
Z: loaded from trap vector
V: loaded from trap vector
C: loaded from trap vector

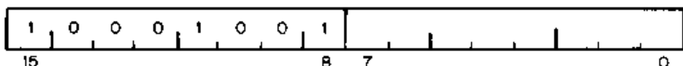
Description:

All operation codes from 104000 to 104377 are EMT instructions and may be used to transmit information to the emulating routine (e.g., function to be performed). The trap vector for EMT is at address 30. The new PC is taken from the word at address 30; the new central processor status (PS) is taken from the word at address 32.

Caution: EMT is used frequently by DIGITAL system software and is therefore not recommended for general use.

TRAP

Trap 104400 to 104777



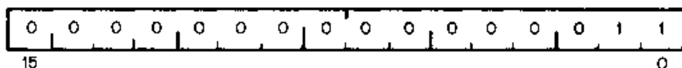
Operation:
↓(SP)←PS
↓(SP)←PC
PC←(34)
PS←(36)

Condition Codes:
N: loaded from trap vector
Z: loaded from trap vector
V: loaded from trap vector
C: loaded from trap vector

Description: Operation codes from 104400 to 104777 are TRAP instructions. TRAPs and EMTs are identical in operation, except that the trap vector for TRAP is at address 34.

Note: Since DEC software makes frequent use of EMT, the TRAP instruction is recommended for general use.

Breakpoint Trap 000003



Operation:

↓(SP)←PS
↓(SP)←PC
PC←(14)
PC←(16)

Condition Codes:

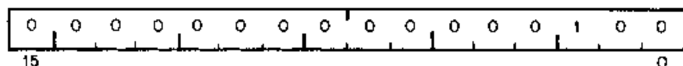
N: loaded from trap vector
Z: loaded from trap vector
V: loaded from trap vector
C: loaded from trap vector

Description:

Performs a trap sequence with a trap vector address of 14. Used to call debugging aids. The user is cautioned against employing code 000003 in programs run under these debugging aids.
(no information is transmitted in the low byte.)

IOT

I/O Trap 000004

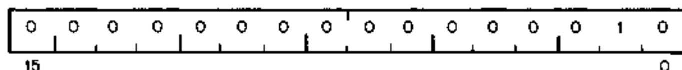


Operation:
↓(SP)←PS
↓(SP)←PC
PC←(20)
PS←(22)

Condition Codes:
N: loaded from trap vector
Z: loaded from trap vector
V: loaded from trap vector
C: loaded from trap vector

Description:
Performs a trap sequence with a trap vector address of 20. Used to call the I/O Executive routine IOX in the paper tape software system, and for error reporting in the Disk Operating System.
(no information is transmitted in the low byte)

Return from Interrupt 000002



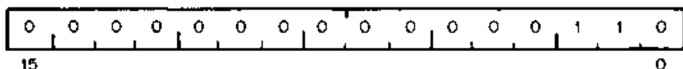
Operation: PC←(SP)↑
PS←(SP)↑

Condition Codes: N: loaded from processor stack
Z: loaded from processor stack
V: loaded from processor stack
C: loaded from processor stack

Description: Used to exit from an interrupt or TRAP service routine. The PC and PS are restored (popped) from the processor stack.

RTT

Return from Trap 000006



Operation: PC←(SP)↑
PS←(SP)↑

Condition Codes: N: loaded from processor stack
Z: loaded from processor stack
V: loaded from processor stack
C: loaded from processor stack

Description: This is the same as the RTI instruction except that it inhibits a trace trap, while RTI permits a trace trap. If a trace trap is pending, the first instruction after the RTT will be executed prior to the next "T" trap. In the case of the RTI instruction the "T" trap will occur immediately after the RTI.

Reserved Instruction Traps—These are caused by attempts to execute instruction codes reserved for future processor expansion (reserved instructions) or instructions with illegal addressing modes (illegal instructions). Order codes not corresponding to any of the instructions described are considered to be reserved instructions. JMP and JSR with register mode destinations are illegal instructions. Reserved and illegal instruction traps occur as described under EMT, but trap through vectors at addresses 10 and 4 respectively.

Stack Overflow Trap
Bus Error Traps—Bus Error Traps are:

1. Boundary Errors—attempts to reference instructions or word operands at odd addresses.

2. Time-Out Errors—attempts to reference addresses on the bus that made no response within 5 μs in the PDP-11/70. In general, these are caused by attempts to reference non-existent memory, and attempts to reference non-existent peripheral devices.

Bus error traps cause processor traps through the trap vector address 4.

Trace Trap—Trace Trap enables bit 4 of the PS and causes processor traps at the end of instruction executions. The instruction that is executed after the instruction that set the T-bit will proceed to completion and then cause a processor trap through the trap vector at address 14. Note that the trace trap is a system debugging aid and is transparent to the general programmer.

The following are special cases and are detailed in subsequent paragraphs.

1. The traced instruction cleared the T-bit.
2. The traced instruction set the T-bit.
3. The traced instruction caused an instruction trap.
4. The traced instruction caused a bus error trap.
5. The traced instruction caused a stack overflow trap.
6. The process was interrupted between the time the T-bit was set and the fetching of the instruction that was to be traced.
7. The traced instruction was a WAIT.
8. The traced instruction was a HALT.
9. The traced instruction was a Return from Trap.

Note: The traced instruction is the instruction after the one that sets the T-bit.

An instruction that cleared the T-bit—Upon fetching the traced instruction an internal flag, the trace flag, was set. The trap will still occur at the end of execution of this instruction. The stacked status word, however, will have a clear T-bit.

An instruction that set the T-bit—Since the T-bit was already set, setting it again has no effect. The trap will occur.

An instruction that caused an Instruction Trap—The instruction trap is sprung and the entire routine for the service trap is executed. If the service routine exits with an RTI or in any other way restores the stacked status word, the T-bit is set again, the instruction following the traced instruction is executed and, unless it is one of the special cases noted above, a trace trap occurs.

An instruction that caused a Bus Error Trap—This is treated as an Instruction Trap. The only difference is that the error service is not as likely to exit with an RTI, so that the trace trap may not occur.

An instruction that caused a stack overflow—The instruction completes execution as usual—the Stack Overflow does not cause a trap. The Trace Trap Vector is loaded into the PC and PS, and the old PC and PS are pushed onto the stack. Stack Overflow occurs again, and this time the trap is made.

An interrupt between setting of the T-bit and fetch of the traced instruction—The entire interrupt service routine is executed and then the T-bit is set again by the exiting RTI. The traced instruction is executed (if there have been no other interrupts) and, unless it is a special case noted above, causes a trace trap.

Note that interrupts may be acknowledged immediately after the loading of the new PC and PS at the trap vector location. To lock out all interrupts, the PS at the trap vector should raise the processor priority to level 7.

A WAIT—The trap occurs immediately.

A HALT—The processor halts. When the continue key on the console is pressed, the instruction following the HALT is fetched and executed. Unless it is one of the exceptions noted above, the trap occurs immediately following execution.

A Return from Trap—The return from trap instruction either clears or sets the T-bit. It inhibits the trace trap. If the T-bit was set and RTT is the traced instruction the trap is delayed until completion of the next instruction.

Power Failure Trap—is a standard PDP-11 feature. Trap occurs whenever the AC power drops below 95 volts or outside 47 to 63 Hertz. Two milliseconds are then allowed for power down processing. Trap vector for power failure is at locations 24 and 26.

Trap priorities—in case multiple processor trap conditions occur simultaneously the following order of priorities is observed (from high to low):

1. Parity error
2. Memory Management violation
3. Stack Limit Yellow
4. Power Failure
5. Floating Point
6. Program Interrupt Request
7. Bus Request
8. Trace Trap

The details on the trace trap process have been described in the trace trap operational description which includes cases in which an instruction being traced causes a bus error, instruction trap, or a stack overflow trap.

If a bus error is caused by the trap process handling instruction traps, trace traps, stack overflow traps, or a previous bus error, the processor is halted.

If a stack overflow is caused by the trap process in handling bus errors, instruction traps, or trace traps, the process is completed and then the stack overflow trap is sprung.

4.7 MISCELLANEOUS

HALT

WAIT

RESET

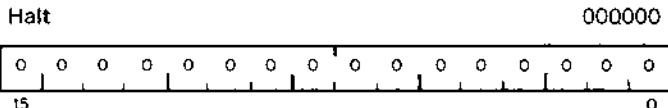
MTPD

MTPI

MFPD

MFPI

Halt 000000



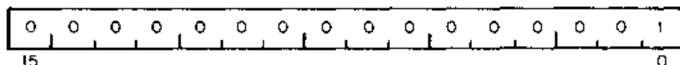
Condition Codes: not affected

Description: Causes the processor operation to cease. The console is given control of the bus. The console data lights display the contents of R0; the console address lights display the address after the halt instruction. Transfers on the UNIBUS are terminated immediately. The PC points to the next instruction to be executed. Pressing the continue key on the console causes processor operation to resume. No INIT signal is given.

Note: A halt issued in Supervisor or User Mode will generate a trap.

WAIT

Wait for Interrupt 000001

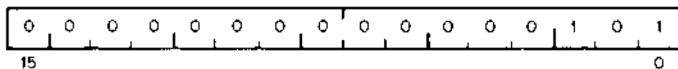


Condition Codes: not affected

Description: Provides a way for the processor to relinquish use of the bus while it waits for an interrupt. Having been given a WAIT command, the processor will not compete for bus use by fetching instructions or operands from memory. This permits higher transfer rates between a device and memory, since no processor-induced latencies will be encountered by bus requests from the device. In WAIT, as in all instructions, the PC points to the next instruction following the WAIT operation. Thus when the service routine executes an RTI instruction, at the end of the routine, the program will resume at the instruction following the WAIT. Note also that Floating Point, Power Fail, and Parity Traps will cause the processor to fall through the WAIT loop.

RESET

Reset External Bus 000005

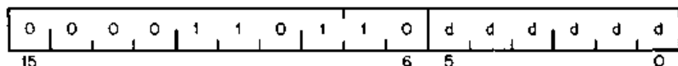


Condition Codes: not affected

Description: Sends INIT on the UNIBUS for 10 ms. All devices on the UNIBUS are reset to their state at power up.

MTPI

Move to Previous Instruction Space 0066DD

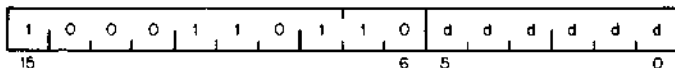


Operation: (temp)←(SP)↑
(dst)←(temp)

Condition Codes: N: set if the source <0; otherwise cleared
Z: set if the source =0; otherwise cleared
V: cleared
C: unaffected

Description: The address of the destination operand is determined in the current address space. MTPI then pops a word off the current stack and stores that word in the destination address in the previous mode's I space (bits 13, 12 of PS).

Move to Previous Data Space 1066DD



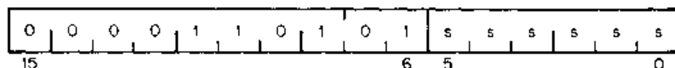
Operation: (temp) ← (SP)↑
(dst) ← (temp)

Condition Codes:
N: set if the source <0; otherwise cleared
Z: set if the source =0; otherwise cleared
V: cleared
C: unaffected

Description: The address of the destination operand is determined in the current address space as in MTPI. MTPD then pops a word off the current stack and stores that word in the destination address in the previous mode's D space.

MFPI

Move from Previous Instruction Space 0065SS



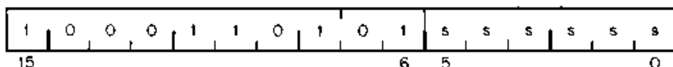
Operation: (temp) ← (src)
↓(SP)←(temp)

Condition Codes:
N: set if the source <0; otherwise cleared
Z: set if the source $=0$; otherwise cleared
V: cleared
C: unaffected

Description: This instruction is provided in order to allow inter-address space communication when the PDP11/45 is using the Memory Management unit. The address of the source operand is determined in the current address space. That is, the address is determined using the SP and memory pages determined by PS<15:14>. The address itself is then used in the previous I space (as determined by PS<13:12> to get the source operand. This operand is then pushed onto the current R6 stack.

MFPD

Move from Previous Data Space 1065SS



Operation: (temp)←(src)
↓(SP)←(temp)

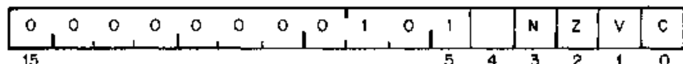
Condition Codes: N: set if the source <0; otherwise cleared
Z: set if the source =0; otherwise cleared
V: cleared
C: unaffected

Description: This instruction is provided in order to allow inter-address space communication when the PDP-11/45 is using the Memory Management unit. The address of the source operand is determined in the current address space. That is, the address is determined using the SP and memory pages determined by PS<15:14>. The address itself is then used in the previous D space (as determined by PS<13:12> to get the source operand. This operand is then pushed on to the current R6 stack.

4.8 Condition Code Operators

CLN SEN
CLZ SEZ
CLV SEV
CLC SEC
CCC SCC

Condition Code Operators 0002XX



Description: Set and clear condition code bits. Selectable combinations of these bits may be cleared or set together. Condition code bits corresponding to bits in the condition code operator (Bits 0-3) are modified according to the sense of bit 4, the set/clear bit of the operator. i.e. set the bit specified by bit 0, 1, 2 or 3, if bit 4 is a 1. Clear corresponding bits if bit 4 = 0.

Mnemonic Operation		OP Code
CLC	Clear C	000241
CLV	Clear V	000242
CLZ	Clear Z	000244
CLN	Clear N	000250
SEC	Set C	000261
SEV	Set V	000262
SEZ	Set Z	000264
SEN	Set N	000270
SCC	Set all CC's	000277
CCC	Clear all CC's	000257
	Clear V and C	000243
	No Operation	000240

Combinations of the above set or clear operations may be ORed together to form combined instructions.

CHAPTER 5

PROCESSOR CONTROL

5.1 GENERAL

This chapter provides detailed information on:

a) CPU registers: CPU Error
System Size
System Identification
Microprogram Break
Processor Status

b) Processor Traps

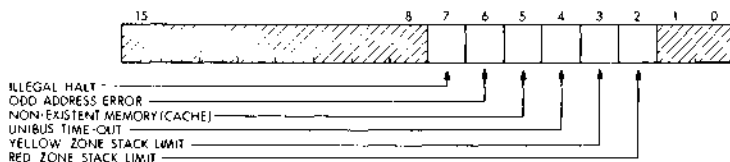
c) Stack Limit

d) Program Interrupt Request

5.2 REGISTERS

The following 5 CPU registers are not accessible from the UNIBUS. They are accessed by program or console control.

CPU Error Register 17 777 766



This register identifies the source of the abort or trap that used the vector at location 4.

BIT	NAME	FUNCTION
7	Illegal Halt	Set when trying to execute a HALT instruction when the CPU is in User or Supervisor mode (not Kernel).
6	Odd Address Error	Set when a program attempts to do a word reference to an odd address.
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.
4	UNIBUS Timeout	Set when there is no response on the UNIBUS within approx. 10 μsec.

BIT	NAME	FUNCTION
3	Yellow Zone Stack Limit	Set when a yellow zone trap occurs.
2	Red Zone Stack Limit	Set when a red zone trap occurs.

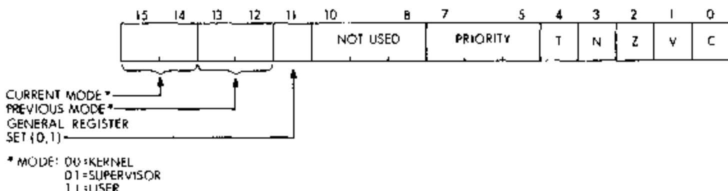
Lower Size Register 17 777 760
This read only register specifies the memory size of the system. It is defined to indicate the last addressable block of 32 words in memory (bit 0 is equivalent to bit 6 of the Physical Address).

Upper Size Register 17 777 762
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.

System I/D Register 17 777 764
This read only register contains information uniquely identifying each system.

Microprogram Break Register 17 777 770
This register is used for maintenance purposes only. It is used with maintenance equipment to provide timing synchronization and testing facilities.

Processor Status Word 17 777 776



The Processor Status Word contains information on the current status of the CPU. This information includes the register set currently in use; current processor priority; current and previous operational modes; the condition codes describing the results of the last instruction; and an indicator for detecting the execution of an instruction to be trapped during program debugging.

5.3 PROCESSOR TRAPS

There are a series of errors and programming conditions which will cause the Central Processor to trap to a set of fixed locations. These include Power Failure, Odd Addressing Errors, Stack Errors, Timeout Errors, Non-Existent Memory References, Memory Parity Errors, Memory Management Violations, Floating Point Processor Exception Traps, use of Reserved Instructions, use of the T bit in the Processor Status Word, and use of the IOT, EMT, and TRAP instructions.

Power Failure
Whenever AC power drops below 95 volts for 110v power (190 volts for 220v) or outside a limit of 47 to 63 Hz, as measured by DC power, the

power fail sequence is initiated. The Central Processor automatically traps to location 24 and the power fail program has 2 msec. to save all volatile information (data in registers), and to condition peripherals for power fail.

When power is restored the processor traps to location 24 and executes the power up routine to restore the machine to its state prior to power failure.

Odd Addressing Errors
This error occurs whenever a program attempts to execute a word instruction on an odd address (in the middle of a word boundary). The instruction is aborted and the CPU traps through location 4.

Time-out Error
This error occurs when a Master Synchronization pulse is placed on the UNIBUS and there is no slave pulse within 10 μsec. This error usually occurs in attempts to address non-existent memory or peripherals.

The offending instruction is aborted and the processor traps through location 4.

Non-Existent Memory Errors
This error occurs when a program attempts to reference a memory address that is larger than indicated by the system size register. The cycle is aborted and the processor traps through vector 4.

Reserved Instructions
There is a set of illegal and reserved instruction which cause the processor to trap through Location 10. The set is fully described in Appendix A.

Trap Handling
Appendix A includes a list of the reserved Trap Vector locations, and System Error Definitions which cause processor traps. When a trap occurs, the processor follows the same procedure for traps as it does for interrupts (saving the PC and PS on the new Processor Stack etc. . . .).

In cases where traps and interrupts occur concurrently, the processor will service the conditions according to the priority sequence illustrated following.

Trap Priorities

Parity error

Memory Management violation

Stack Limit Yellow

Power Failure (power down)

Floating Point exception trap

Program Interrupt Request (PIR) level 7

Bus Request (BR) level 7

PIR 6

BR 6

PIR 5

BR 5

PIR 4

BR 4

PIR 3

PIR 2

PIR 1

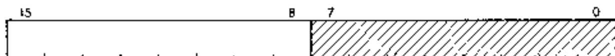
Trace trap

5.4 STACK LIMIT

The Stack Limit allows program control of the lower limit for permissible stack addresses. This limit may be varied in increments of $(400)_8$ bytes or $(200)_8$ words, up to a maximum address of 177 400 (almost the top of a 32K memory).

The normal boundary for stack addresses is 400. The Stack Limit option 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.

There is a Stack Limit Register, with the following format:



The Stack Limit Register can be addressed as a word at location 17 777774, or as a byte at location 17 777775. The register is accessible to the processor and console, but not to any bus device.

The 8 bits, 15 through 8, contain the stack limit information. 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 (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 effected. 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 stack is repositioned 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 Stack Limit Register (SL) are compared to the stack address to determine if a violation has occurred. The least significant

bit of the register (bit 8) has a value of $(400)_8$. The determination of the violation zones is as follows:

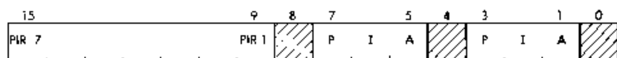
Yellow Zone = (SL) + $(340 \text{ through } 377)_8$	execute, then trap
Red Zone ≤ (SL) + $(337)_8$	abort, then trap to location 4

If the Stack Limit Register contents were zero:

Yellow Zone = 340 through 377
Red Zone = 000 through 337

5.5 PROGRAM INTERRUPT REQUESTS

A request is booked by setting one of the bits 15 through 9 (for PIR 7—PIR 1) in the Program Interrupt Register at location 17 777772. The hardware sets bits 7—5 and 3—1 to the encoded value of the highest PIR bit set. This Program Interrupt Active (PIA) should be used to set the Processor Level and also index through a table of interrupt vectors for the seven software priority levels. The Figure shows the layout of the PIR Register.



Program Interrupt Request Register

When the PIR is granted, the Processor will Trap to location 240 and pick up 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.

The actual interrupt dispatch program should look like:

```
MOVB PIR,PS          ; places Bits 5—7 in PSW Priority Level
                     ; Bits
MOV  R5,—(SP)        ; save R5 on the stack
MOV  PIR,R5
BIC  #177761,R5      ; Gets Bits 1—3
JMP  @DISPAT(R5)     ; use to index through table
                     ; which requires 15 core locations.
```

CHAPTER 6

ADDRESSING

6.1. GENERAL

This chapter provides detailed information on:

a) Address space
b) Memory management
c) UNIBUS Map
d) Non-existent memory errors

6.2 ADDRESS SPACE

There are 3 separate address spaces used:

a) 16 bits, program virtual space
b) 18 bits, UNIBUS space
c) 22 bits, physical space

A 22-bit physical address references a unique core memory location (or register). The UNIBUS Map performs the conversion of 18-bit UNIBUS addresses to 22-bit physical addresses. Within the CPU, the Memory Management unit converts 16-bit program virtual addresses to 22-bit physical addresses. Registers within these two memory extension units are used in conjunction with the virtual or UNIBUS address to produce the physical address. See Figure 6-1.

CPU Addresses

Of the over 2 million 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 (1 word = 2 bytes). If the CPU address is between 00 000 000 and 16 777 777, an attempt is made to reference physical memory. If the address is in the top 128K, 17 000 000 to 17 777 777, the lower 18 bits of the address are placed on the UNIBUS. See Figure 6-2.

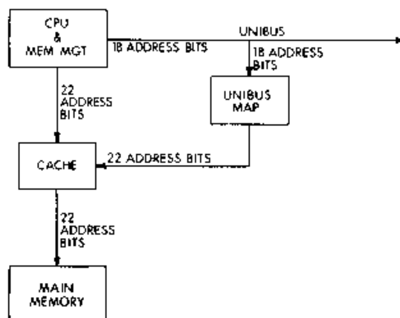


Figure 6-1 Address Paths in the PDP-11/70

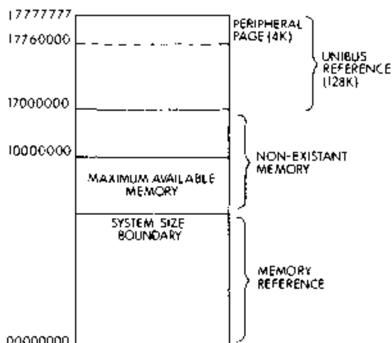


Figure 6-2 Physical Address Space

6.3 CPU MAPPING

Mapping of processor addresses is performed in 1 of 3 possible ways.

16-Bit Mapping
There is fixed relocation 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 to 17 777 777. Refer to Figure 6-3. 16-bit mapping operation occurs after Power Up, Console Start, or the RESET instruction.

18-Bit Mapping
32K virtual addresses for each of the 3 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 to 17 777 777. Refer to Figure 6-4.

22-Bit Mapping
This mode produces full 22-bit addresses for accessing all of PDP-11/70 physical memory. The top 128K addresses cause UNIBUS cycles to addresses 17 000 000 to 17 777 777. Refer to Figure 6-5.

6.4 COMPATIBILITY

Operation with 16-bit and 18-bit mapping can be used such that the computer is compatible with other PDP-11 computers, such as the PDP-11/20 and the PDP-11/45. Operating in this manner means that software written for another PDP-11 can be run on the PDP-11/70 without modification.

Mapping	Mem Mgt	UNIBUS Map Relocation	Compatible With
16 Bit	Off	Off	PDP-11/05, 11/10, 11/15, 11/20
18 Bit	On	Off	PDP-11/35, 11/40, 11/45, 11/50
22 Bit	On	Off or On	PDP-11/70

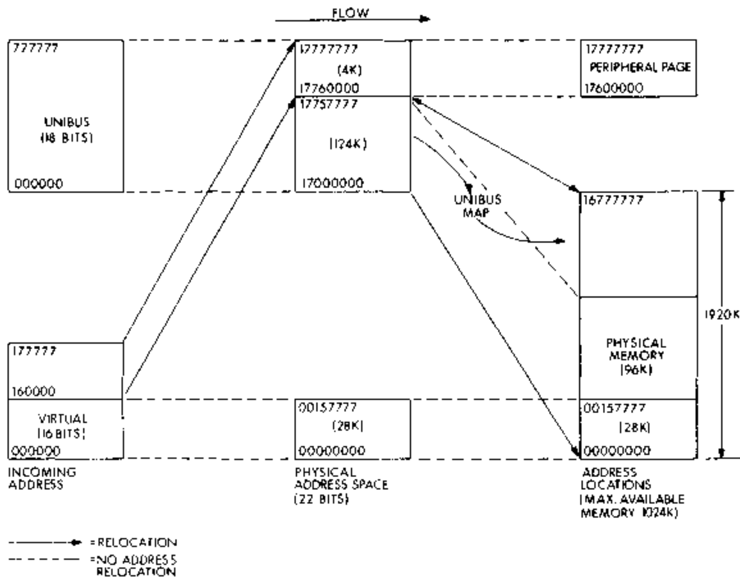


Figure 6-3 16 Bit Mapping

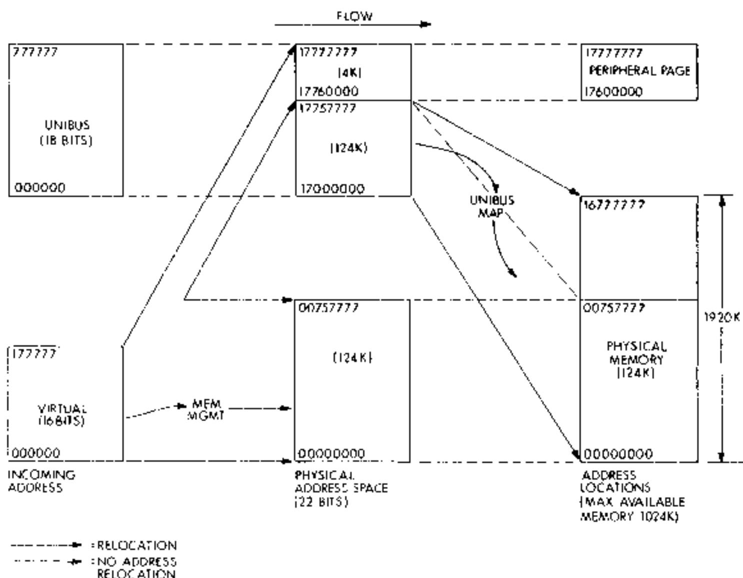


Figure 6-4 18-Bit Mapping

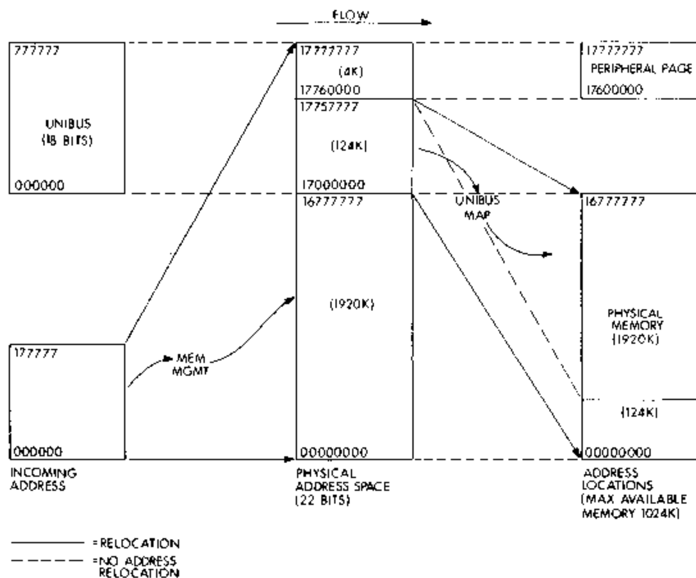


Figure 6-5 22 Bit Mapping

6.5 MEMORY MANAGEMENT

6.5.1 General

The PDP-11/70 Memory Management Unit provides the hardware facilities necessary for complete memory management 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 would be resident in memory at any given time. The task of the supervisory program would be: control the execution of the various user programs, manage the allocation of memory and peripheral device resources, and safeguard the integrity of the system as a whole by careful control of each user program.

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

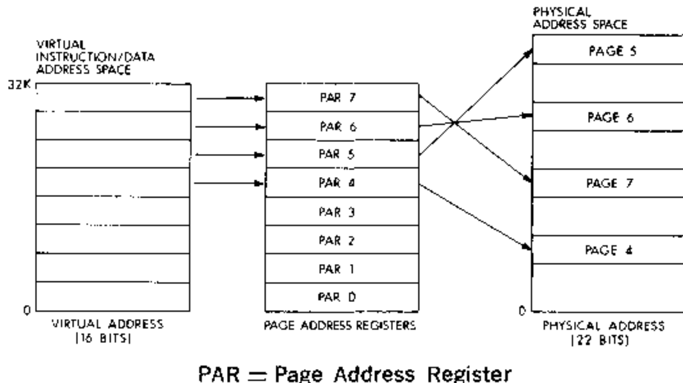
The basic characteristics of the PDP-11/70 Memory Management Unit are:

- 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)

6.5.2 Virtual Addressing

When the PDP-11/70 Memory Management Unit is operating, 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 physical address. The information contained in the Virtual Addess (VA) is combined with relocation information contained in the Page Address Register (PAR) to yield a 22-bit Physical Address (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.



PAR = Page Address Register

Figure 6-6 Virtual Address Mapping into Physical Address

The starting physical address for each page is an integral multiple of 32 words, and each page has a maximum size of 4096 words. Pages may be located anywhere within the Physical Address space. The determination of which set of 16 pages registers is used to form a Physical Address is made by the current mode of operation of the CPU, i.e., Kernel, Supervisor or User mode.

6.5.3 Interrupt Conditions under Memory Management Control

The Memory Management Unit relocates all addresses. Thus, when it is enabled, all trap, abort, and interrupt vectors are considered to be in Kernel mode Virtual Address Space. When a vectored transfer occurs, control is transferred according to a new Program Counter (PC) and Processor Status Word (PS) contained in a two-word vector relocated through the Kernel Page Address Register Set. Relocation of trap addresses means that the hardware is capable of recovering from a failure in the first physical bank of memory.

When a trap, abort, or interrupt occurs the "push" of the old PC, old PS is to the User/Supervisor/Kernel R6 stack specified by CPU mode bits 15,14 of the new PS in the vector (bits 15,14: 00 = Kernel, 01 = Supervisor, 11 = User). The CPU mode bits also determine the new PAR set. In this manner it is possible for a Kernel mode program to have complete control over service assignments for all interrupt conditions, since the interrupt vector is located in Kernel space. The Kernel program may assign the service of some of these conditions to a Supervisor or User mode program by simply setting the CPU mode bits of the new PS in the vector to return control to the appropriate mode.

6.5.4 Construction of a Physical Address

All addresses with memory relocation enabled either reference information in 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 PAR's 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 Physical Address (PA) comes from the Virtual Address (VA), which is illustrated in Figure 6-7, and the appropriate PAR set.

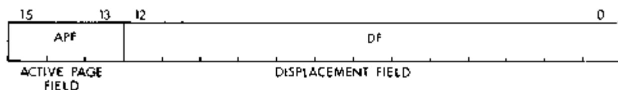


Figure 6-7 Interpretation of a Virtual Address

The Virtual Address (VA) 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 Physical Address (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 6-8.

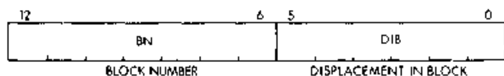


Figure 6-8 Displacement Field of Virtual Address

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 Physical Address 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 Physical Address (PA) is illustrated in Figure 6-9.

The logical sequence involved in constructing a Physical Address (PA) is as follows:

1. Select a set of Page Address Registers depending on the space being referenced.
2. The Active Page Field (APF) of the Virtual Address is used to select a Page Address Register (PAR0-PAR7).
3. The Page Address Field (PAF) of the selected Page Address Register (PAR) contains the starting address of the currently active page as a block number in physical memory.
4. The Block Number (BN) from the Virtual Address (VA) is added to the Page Address Field (PAF) to yield the number of the block in physical memory (PBN-Physical Block Number) which will contain the Physical Address (PA) being constructed.
5. The Displacement in Block (DIB) from the Displacement Field (DF) of the Virtual Address (VA) is joined to the Physical Block Number (PBN) to yield a true 22-bit PDP-11/70 Physical Address (PA).

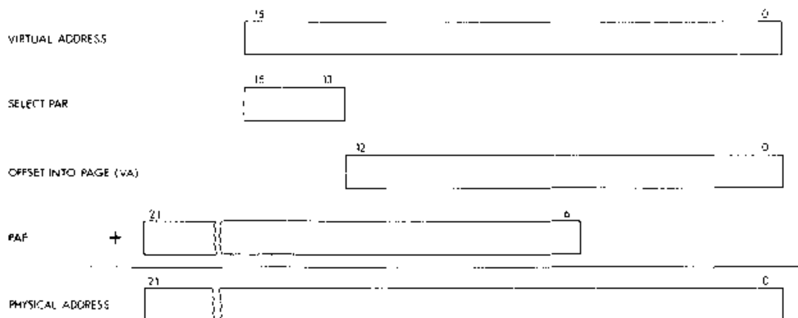


Figure 6-9 Construction of a Physical Address

6.5.5 Management Registers

The PDP-11/70 Memory Management Unit implements three sets of 32 sixteen 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 Proces-

sor 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 Managements Register #3. Each group is further subdivided into two parts of 8 registers. One part is the Page Address Register (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 virtual address. 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 physical address space along with the UNIBUS I/O device registers.

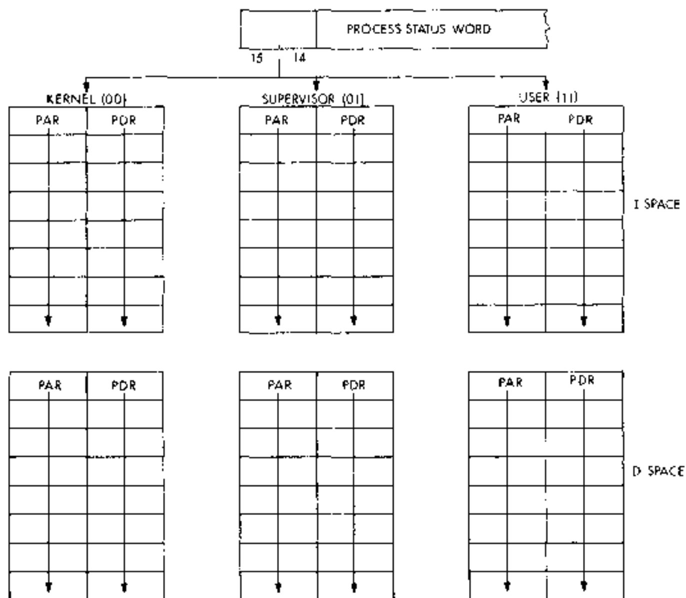


Figure 6-10 Active Page Registers

Page Address Registers (PAR)
The Page Address Register (PAR) contains the Page Address Field (PAF), 16-bit field, which specifies the starting address of the page as a block number in physical memory.

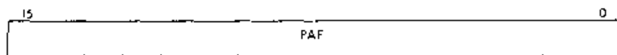


Figure 6-11 Page Address Register

The Page Address Register (PAR) which contains the Page Address Field (PAF) may be alternatively thought of as a relocation register containing a relocation constant, or as a base register containing a base address. Either interpretation indicates the basic importance of the Page Address Register (PAR) as a relocation tool.

Page Descriptor Register
The Page Descriptor Register (PDR) contains information relative to page expansion, page length, and access control.

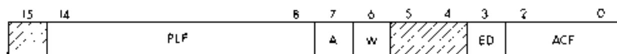


Figure 6-12 Page Description Register

Access Control Field (ACF)
This three-bit field, occupying bits 2-0 of the Page Descriptor Register (PDR) contains the access rights to this particular page. The access codes or "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 fact, when a memory reference causes a trap to occur, the trap does not occur until the entire instruction has been completed. Aborts are used to catch "missing page faults," prevent illegal access, etc.; traps are used as an aid in gathering memory management information.

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. "Write" is synonymous with what is usually called a "store" or "modify" in many computer systems.

The modes 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

It should be noted that the use of I Space provides the user with a further form of protection, execute only.

Access Information Bits
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 be simply overlaid.

Note that A and W bits are "reset" to "0" whenever either PAR or PDR is modified (written into).

Expansion Direction (ED)
Bit 03 of the Page Description Register (PDR) specifies in which direction the page expands. If ED = 0 the page expands upwards from Block Number 0 to include blocks with higher addresses; if ED = 1, the page expands downwards from Block Number 127 to include blocks with lower addresses. Upward expansion is usually used for program space while downward expansion is used for stack space.

Page Length Field (PLF)
This seven-bit field, occupying bits 14-8 of the Page Descriptor Register (PDR), specifies the block number, which defines the boundary of that page. The block number of the Virtual Address is compared against the Page Length Field to detect Length Errors. An error occurs when expanding upwards if the block number is greater than the Page Length Field, and when expanding downwards if the block number is less than the Page Length Field.

Reserved Bits
Bits 15, 5, and 4 are spare and are always read as 0, and should never be written. They are unused and reserved for possible future expansion.

6.5.6 Fault Recovery Registers
Aborts and traps generated by the Memory Management hardware are vectored through Kernel virtual location 250, Memory Management Registers #0, #1, #2 and #3 are used in order to differentiate an abort from a trap, determine why the abort or trap occurred, and allow for easy program restarting. Note that an abort or trap to a location which is itself an invalid address will cause another abort or trap. Thus the Kernel program must insure that Kernel Virtual Address 250 is mapped into a valid address, otherwise a loop will occur which will require console intervention.

Memory Management Register #0 (MMR0) (status and error indicators)
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 6-13.

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. That is, 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.

Bits 15-13 when set (error conditions) cause Memory Management to freeze the contents of bits 1-7 and Memory Management Registers #1 and #2. This has been 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.

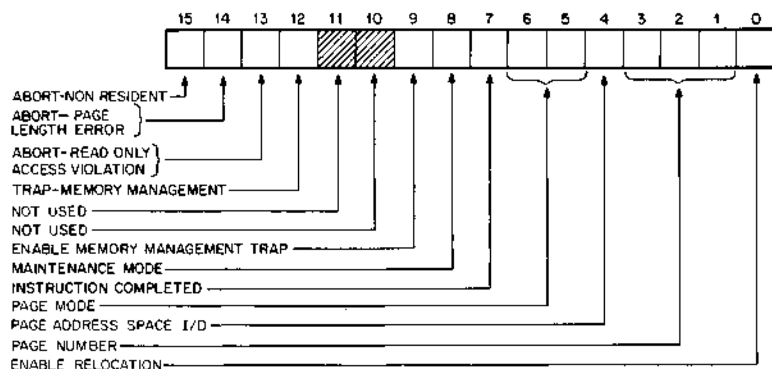


Figure 6-13 Format of Memory Management Register #0 (MMR0)

Abort—Non-Resident, Bit 15
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.

Abort—Page Length, Bit 14
Bit 14 is the "Abort Page Length" bit. It is set by attempting to access a location in a page with a block number (Virtual Address bits, 12-6) that is outside the area authorized by the Page Length Field (PLF) of the Page Descriptor Register (PDR) for that page. Bits 14 and 15 may be set simultaneously by the same access attempt. Bit 14 is also set by attempting to use Memory Relocation with a processor mode of 2.

Abort—Read Only, Bit 13
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.

Trap—Memory Management, Bit 12
Bit 12 is the "Trap—Memory Management" bit. It is set whenever a Memory Management trap condition occurs; that is, a read operation which references a page with an Access Control Field (ACF) of 1 or 4, or a write operation to a page with an ACF key of 4 or 5.

Bits 11, 10
Bits 11 and 10 are spare and are always read as 0, and should never be written. They are unused and reserved for possible future expansion.

Enable Memory Management Traps, Bit 9
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 Memory Management Trap conditions. When bit 9 is set to 1, the next Memory Management trap condition will cause a trap, vectored through Kernel Virtual Address 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 anyway.

Maintenance/Destination Mode, Bit 8
Bit 8 specifies that only destination mode references will be relocated using Memory Management. This mode is only used for maintenance purposes.

Instruction Completed, Bit 7
Bit 7 indicates that the current instruction has been completed. It will be set to 0 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. Bit 7 is Read-Only (it cannot be written). It is initialized to a 1. Note that EMT, TRAP, BPT, and IOT do not set bit 7.

Processor Mode, Bits 5 & 6
Bits 5 and 6 indicate the CPU MODE (Kernel/Supervisor/User) associated with the page causing the abort (Kernel = 00, Supervisor = 01, User = 11, Illegal Mode = 10). If an illegal mode is specified, bits 15 and 14 will be set.

Page Address Space, Bit 4
Bit 4 indicates the type of address space (I or D) the Unit was in when a fault occurred (0 = I Space, 1 = D Space). It is used in conjunction with bits 3-1, Page Number.

Page Number, Bits 3 to 1
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.

Enable Relocation, Bit 0
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.

Memory Management Register #1 (MMR1)
MMR1 records any autoincrement/decrement of the general purpose registers, including explicit references through the PC. MMR1 is cleared at the beginning of each instruction fetch. Whenever a general purpose register is either autoincremented or autodecremented the register number and the amount (in 2s 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 before an "abort-causing" reference. Register numbers are recorded "MOD 8"; 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 as contained in the PS 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.

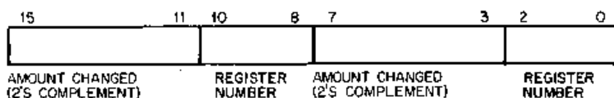


Figure 6-14 Format of Memory Management Register #1 (MMR1)

Memory Management Register #2
MMR2 is loaded with the 16-bit Virtual Address (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 aborts and parity traps. Note that MMR2 does not get the Trap Vector on EMT, TRAP, BPT and IOT instructions. MMR2 is Read-Only; it can not be written. MMR2 is the Virtual Address Program Counter.

Memory Management Register #3
The Memory Management Register #3 (MMR3) enables or disables the use of the D space PAR's and PDR's and 22-bit mapping and UNIBUS 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 clear, 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.

If bit 4 is clear and Memory Management is enabled (bit 0 of MMR0 is set), the computer uses 18-bit mapping. If bit 4 is set and Memory Man-

agement 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 to 15 are unused. On initialization this register is set to 0 and only I space is in use.

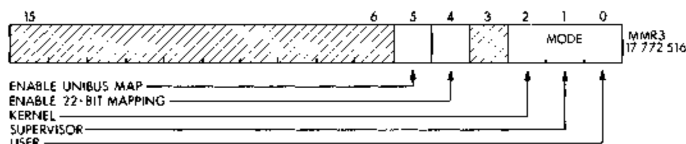


Figure 6-15 Format of Memory Management Register #3 (MMR3)

Bit	State	Operation
5	0	UNIBUS Map relocation disabled
	1	UNIBUS Map relocation enabled
4	0	Enable 18-bit mapping } if bit 0 of MMR0 is set
	1	Enable 22-bit mapping } if bit 0 of MMR0 is set
2	1	Enable Kernel D Space
1	1	Enable Supervisor D Space
0	1	Enable User D Space

Instruction Back-Up/Restart Recovery
The process of "backing-up" and restarting a partially completed instruction involves:

1. Performing the appropriate memory management tasks to alleviate the cause of the abort (e.g., loading a missing page, etc.)
2. Restoring the general purpose registers indicated in MMR1 to their original contents at the start of the instruction by subtracting the "modify value" specified in MMR1.
3. Restoring the PC to the "abort-time" PC by loading R7 with the contents of MMR2, which contains the value of the Virtual PC at the time the "abort-generating" instruction was fetched.

Note that this back-up/restart procedure assumes that the general purpose register used in the program segment will not be used by the abort recovery routine. This is automatically the case if the recovery program uses a different general register set.

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.

Multiple Faults
Once an abort has occurred, any subsequent errors that occur will not affect the state of the machine. The information saved in MMRO thru

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.

6.5.7 Examples

Normal Usage

The Memory Management Unit provides a very general purpose memory management tool. It can be used in a manner as simple or complete as desired. It can be anything from a simple memory expansion device to a very complete memory management facility.

The variety of possible and meaningful ways to utilize the facilities offered by the Memory Management Unit means that both single-user and multi-programming systems have complete freedom to make whatever memory management decisions best suit their individual needs. Although a knowledge of what most types of computer systems seek to achieve may indicate that certain methods of utilizing the Memory Management Unit will be more common than others, there is no limit to the ways to use these facilities.

In most normal applications, it is assumed that the control over the actual memory page assignments and their protection resides in a supervisory type program which would operate at the nucleus of a CPU's executive (Kernel) mode. It is further assumed that this Kernel mode program would set access keys in such a way as to protect itself from willful or accidental destruction by other Supervisor mode or User mode programs. The facilities are also provided such that the nucleus can dynamically assign memory pages of varying sizes in response to system needs.

Typical Memory Page

When the Memory Management Unit is enabled, the Kernel mode program, a Supervisor mode program and a User mode program each have eight active pages described by the appropriate Page Address Registers and Page Descriptor Registers for data, and eight, for instructions. Each segment is made up of from 1 to 128 blocks and is pointed to by the Page Address Field (PAF) of the corresponding Page Address Register (PAR) as illustrated in Figure 6-16.

The memory segment illustrated in Figure 6-16 has the following attributes:

1. Page Length: 40 blocks.
2. Virtual Address Range: 140000—144777.
3. Physical Address Range: 312000—316777.
4. No trapped access has been made to this page.
5. Nothing has been modified (i.e. written) in this page.
6. Read-Only Protection.
7. Upward Expansion.

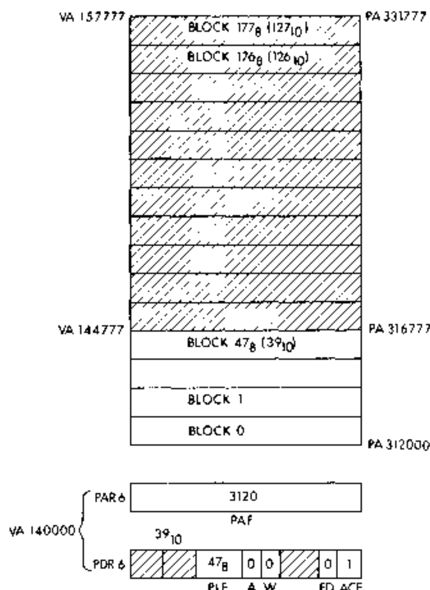


Figure 6-16 Typical Memory Page

These attributes were determined according to the following scheme:

1. Page Address Register (PAR6) and Page Descriptor Register (PDR6) were selected by the Active Page Field (APF) of the Virtual Address (VA). (Bits 15-13 of the VA $= 6_8$.)
2. The initial address of the page was determined from the Page Address Field (PAF) of PAR6 ($312000 = 3120_8$ blocks x 40_8 (32_{10}) words per block x 2 bytes per word).

   Note that the PAR which contains the PAF constitutes what is often referred to as a base register containing a base address or a relocation register containing relocation constant.
3. The page length ($47_8 + 1 = 40_{10}$ blocks) was determined from the Page Length Field (PLF) contained in Page Descriptor Register PDR6. Any attempts to reference beyond these 40_{10} blocks in this page will cause a "Page Length Error," which will result in an abort, vectored through Kernel Virtual Address 250.
4. The Physical Addresses were constructed according to the scheme illustrated in Figure 6-9.
5. The Access bit (A-bit) of PDR6 indicates that no trapped access has been made to this page (A bit = 0). When an illegal or trapped reference, (i.e. a violation of the Protection Mode specified by the Access Control Field (ACF) for this page), or a trapped reference (i.e. Read in this case), occurs, the A-bit will be set to a 1.

6. The Written bit (W-bit) indicates that no locations in this page have been modified (i.e. written). If an attempt is made to modify any location in this particular page, an Access Control Violation Abort will occur. If this page were involved in a disk swapping or memory overlay scheme, the W-bit would be used to determine whether it had been modified and thus required saving before overlay.
7. This page is Read-Only protected; i.e. no locations in this page may be modified. In addition, a memory management trap will occur upon completion of a read access. The mode of protection was specified by the Access Control Field (ACF) of PDR6.
8. The direction of expansion is upward (ED = 0). If more blocks are required in this segment, they will be added by assigning blocks with higher relative addresses.

Note that the various attributes which describe this page can all be determined under software control. The parameters describing the page are all loaded into the appropriate Page Address Register (PAR) and Page Descriptor Register (PDR) under program control. In a normal application it is assumed that the particular page which itself contains these registers would be assigned to the control of a supervisory type program operating in Kernel mode.

Non-Consecutive Memory Pages
It should be noted at this point that although the correspondence between Virtual Addresses (VA) and PAR/PDR pairs is such that higher VAs have higher PAR/PDR's, this does not mean that higher Virtual Addresses (VA) necessarily correspond to higher Physical Addresses (PA). lit is quite simple to set up the Page Address Fields (PAF) of the PAR's in such a way that higher Virtual Address blocks may be located in lower Physical Address blocks as illustrated in Fig. 6-17.

Note that although a single memory page must consist of a block of contiguous locations, memory pages as macro units do not have to be located in consecutive Physical Address (PA) locations. It also should be realized that the assignment of memory pages is not limited to consecutive non-overlapping Physical Address (PA) locations.

Stack Memory Pages
When constructing PDP-11/70 programs it is often desirable to isolate all program variables from "pure code" (i.e. program instructions) by placing them on a register indexed stack. These variables can then be "pushed" or "popped" from the stack area as needed (see Chapter 3. Addressing Modes). Since all PDP-11 Family stacks expand by adding locations with lower addresses, when a memory page which contains "stacked" variables needs more room it must "expand down," i.e. add blocks with lower relative addresses to the current page. This mode of expansion is specified by setting the Expansion Direction (ED) bit of the appropriate Page Descriptor Register (PDR) to a 1. Figure 6-18 illustrates a typical "stack" memory page. This page will have the following parameters:

PAR6: PAF = 3120

PDR6: PLF = 175_8 or 125_{10} (128_{10}-3)

ED = 1

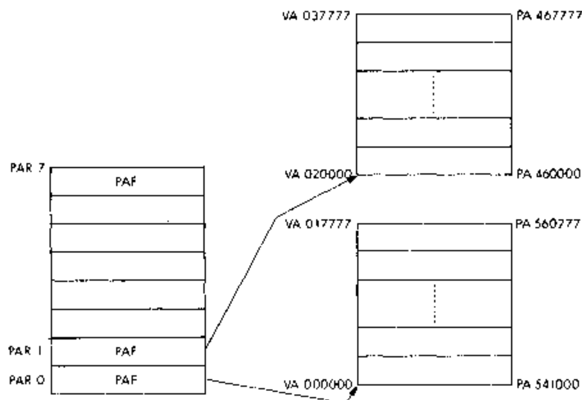


Figure 6-17 Non-Consecutive Memory Pages

A = 0 or 1

W = 0 or 1

ACF = nnn (to be determined by programmer as the need dictates).

note: the A, W bits will normally be set by hardware.

In this case the stack begins 128 blocks above the relative origin of this memory page and extends downward for a length of three blocks. A "PAGE LENGTH ERROR" abort vectored through Kernel Virtual Address (VA) 250 will be generated by the hardware when an attempt is made to reference any location below the assigned area, i.e. when the Block Number (BN) from the Virtual Address (VA) is less than the Page Length Field (PLF) of the appropriate Page Descriptor Register (PDR).

6.5.8 Transparency

It should be clear at this point that in a multiprogramming application it is possible for memory pages to be allocated in such a way that a particular program seems to have a complete 32K basic PDP-11/70 memory configuration. Using Relocation, a Kernel Mode supervisory-type program can easily perform all memory management tasks in a manner entirely transparent to a Supervisor or User mode program. In effect, a PDP-11/70 System can utilize its resources to provide maximum throughput and response to a variety of users each of which seems to have a powerful system "all to himself."

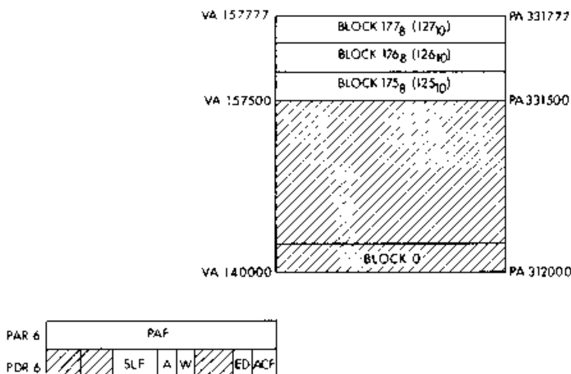


Figure 6-18 Typical Stack Memory Page

6.5.9 Memory Management Unit—Register Map

REGISTER	ADDRESS
Memory Mgt Register #0(MMR0)	**17 777572**
Memory Mgt Register #1(MMR1)	**17 777574**
Memory Mgt Register #2(MMR2)	**17 777576**
Memory Mgt Register #3(MMR3)	**17 772516**
User I Space Descriptor Register (UISDR0)	**17 777600**
.	.
.	.
.	.
User I Space Descriptor Register (UIDR7)	**17 777616**
User D Space Descriptor Register (UDSDR0)	**17 777620**
.	.
.	.
.	.
User D Space Descriptor Register (UDSDR7)	**17 777636**
User I Space Address Register (UISAR0)	**17 777640**
.	.
.	.
.	.
User I Space Address Register (UISAR7)	**17 777656**
User D Space Address Register (UDSAR0)	**17 777660**
.	.
.	.
.	.
User D Space Address Register (UDSAR7)	**17 777676**
Supervisor I Space Descriptor Register (SISDR0)	**17 772200**
.	.
.	.
.	.
Supervisor I Space Descriptor Register (SISDR7)	**17 772216**

REGISTER	ADDRESS
Supervisor D Space Descriptor Register (SDDR0)	17 772226
.	.
.	.
.	.
Supervisor D Space Descriptor Register (SDSDR7)	17 772236
Supervisor I Space Address Register (SISAR0)	17 772240
.	.
.	.
.	.
Supervisor I Space Address Register (SISAR7)	17 772256
Supervisor D Space Address Register (SDSAR0)	17 772260
.	.
.	.
.	.
Supervisor D Space Address Register (SDSDR7)	17 772276
Kernel I Space Descriptor Register (KISDR0)	17 772300
.	.
.	.
.	.
Kernel I Space Descriptor Register (KIDSR7)	17 772316
Kernel D Space Descriptor Register (KDSDR0)	17 772320
.	.
.	.
.	.
Kernel D Space Descriptor Register (KDSDR7)	17 772336
Kernel I Space Address Register (KISAR0)	17 772340
.	.
.	.
.	.
Kernel I Space Address Register (KISAR7)	17 772356
Kernel D Space Address Register (KDSAR0)	17 772360
.	.
.	.
.	.
Kernel D Space Address Register (KDSAR7)	17 772376

6.6 UNIBUS MAP

The UNIBUS Map performs the conversion that allows devices on the UNIBUS to communicate with physical memory by means of Non-Processor Requests (NPR's). 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 registers and is called the Peripherals Page; see Figure 6-19. The lower 124K addresses are used by the UNIBUS Map to reference physical memory.

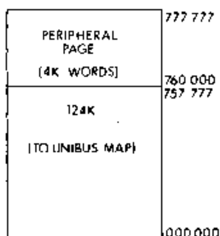


Figure 6-19 UNIBUS Address Space

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. 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; see Figure 6-20. These registers have UNIBUS addresses in the range 770 200 to 770 372.

If UNIBUS Map relocation is enabled, the 5 high order bits of the UNIBUS address are used to select one of the 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; see Figure 6-21. To form the physical address, the 13 low order bits of the UNIBUS address are added to 22 bits of the selected mapping register to produce the 22-bit physical address. Refer to Figure 6-22. The lowest order bit of all mapping registers is always a zero, since relocation is always on word boundaries.

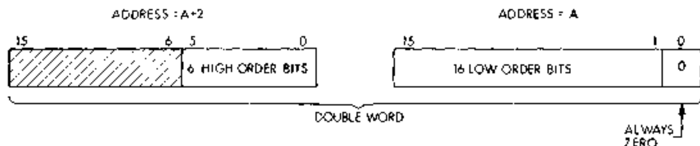


Figure 6-20 Single Mapping Register (1 of 31)

6.7 NON-EXISTENT MEMORY ERRORS

After a 22-bit physical address is generated, the CPU looks at the 4 high order bits, bits 18 to 21, to see if they are all ONES. If this is true (range 17 000 000 to 17 17 777 777), the lower 18 bits are used for a UNIBUS address. If after 10 to 20 μsec, there is no response, the CPU does a UNIBUS Timeout abort, and bit 4 in the CPU error Register is set.

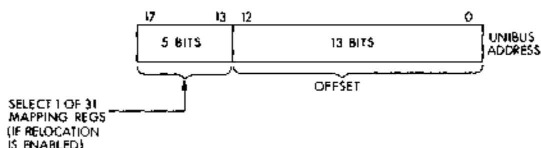


Figure 6-21 18-bit UNIBUS Address

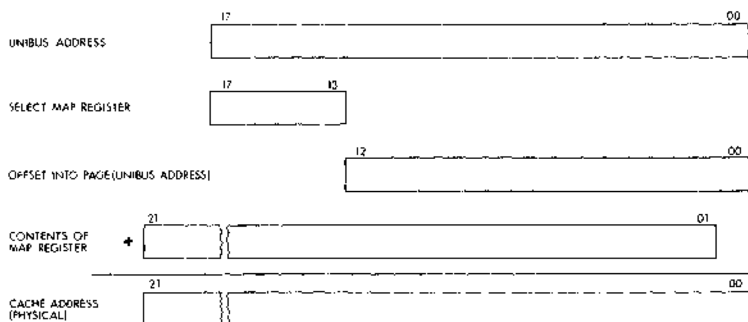


Figure 6-22 Construction of a Physical Address

If the 4 high order bits are not all ONES, the address is compared against the System Size register. If the physical address is higher than the amount of implemented physical memory, the CPU does an immediate non-existent memory abort, and bit 5 in the CPU Error Register is set. Note that it is not necessary to do a time-out, since the maximum physical memory on the system is indicated in the System Size register.

When memory is accessed from the UNIBUS via the UNIBUS Map, a memory cycle is requested. If the memory location is not in physical memory, the memory bus times out . Since there is no response on the UNIBUS, the UNIBUS master also times-out.

CHAPTER 7

MEMORY SYSTEM

7.1 GENERAL

This chapter provides detailed information on:

a) Memory system
b) Cache memory
c) Main memory
d) Parity

An overall block diagram of the PDP-11/70 is shown in Figure 7-1. From a functional standpoint, main memory and the cache can be treated as a single unit of memory.

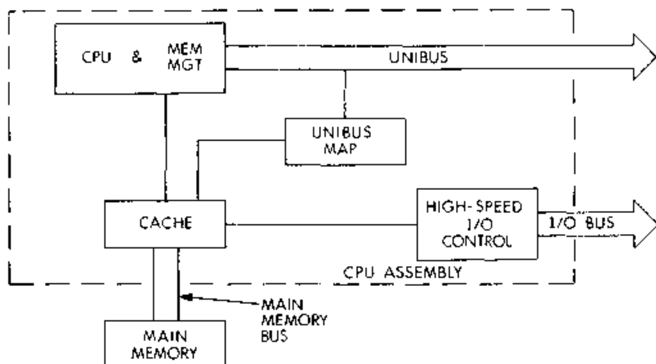


Figure 7-1 Block Diagram of PDP-11/70

7.2 CACHE MEMORY

7.2.1 Introduction

A cache memory is a small, high-speed memory that maintains a copy of automatically selected portions of main memory for faster access to instructions and data. A computer system, using a cache memory, appears the same as a conventional system with core memory, except that the execution of programs is noticeably faster. The only difference is in system timing; there are no changes in programming! The operation is transparent to the user.

Figure 7-2 shows the block diagram for a system with cache memory. Main memory is replaced by a combination of cache memory plus main memory. The cache system simulates a system having a large amount of fast memory. The cache itself uses a small amount of very fast semi-

conductor memory; the main memory uses slower core memory. The key to the effectiveness of a cache is the algorithm which automatically and dynamically allocates (transfers) the data most needed to the fast memory.

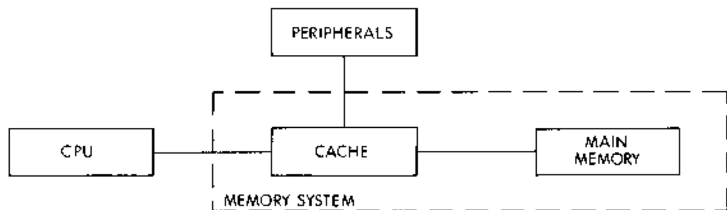


Figure 7-2 Memory System

The statistics of program behavior make a cache system work. All of the data is stored in main memory; a copy of some of the data is stored in the cache. If most of the time the needed data is in the fast memory, the program will execute quickly, slowing down only when accesses must be made to main memory. Other semiconductor-core systems attempt to achieve this goal by having the programmer guess ahead of time which sections of the program should go in which memory. The cache system achieves 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.

A cache memory predicts which words a program will most probably require soon. The principle of program locality states that programs have a tendency to make most accesses in the neighborhood of locations accessed in the recent past. Programs typically execute instructions in straight lines or small loops, with the next few accesses likely to be within a few words ahead or behind of the current location. Stacks grow and shrink from one end, with the next few accesses near the current top. Data elements are often scanned through sequentially. The cache makes use of this type of program behavior by bringing in extra words on each access to main memory (look ahead) and keeping copies of recently used words (look behind).

From a cost effectiveness stnadpoint, a cache system offers faster system speed for the cost of only a small quantity of fast memory plus associated logic. How much faster depends on the size and organization of the cache not on the size of main memory. The user receives a very substantial speed improvement for a modest cost, and there are no programming changes. Although the exact speed improvement depends on the particular program, a judicious choice of architecture and algorithm will produce good results for useful programs.

The fundamental concern is execution speed. This is affected by the speeds of fast and slow memory and by the percentage of times memory references will find the data within the cache and therefore allow faster execution. When the needed data is found in the cache, a hit is said to occur. A miss occurs when the data is not in the cache.

7.2.2 The PDP-11/70 Cache

The architecture of the cache chosen for the PDP-11/70 is described in this section. It represents a carefully thought out approach, backed by extensive program simulations to determine hit statistics. Figure 7-1 shows the basic block diagram of the PDP-11/70 memory system. The size of the cache memory is 1,024 words (2,048 bytes), organized as a two-way associative cache with two-word blocks. This means there are two groups in the cache; each group contains 256 blocks of data, and each block contains two PDP-11 words (see Figures 7-3 & 7-4). Each block also has a tag field, which contains information to construct the address in main memory where the original copy of this data block resides. The data from main memory can be stored within the cache in one index position determined by its physical address. Refer to Figure 7-5 for the organization of the 22-bit physical address. The 8-bit index field (bits 2 to 9) determine which element of the array will contain the data (but it can be in either Group 0 or Group 1).

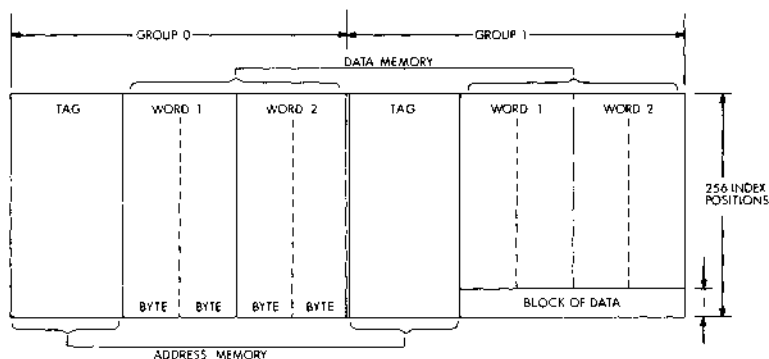


Figure 7-3 Cache Memory (1024 words)

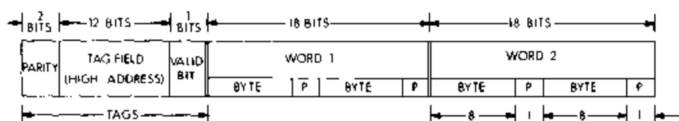


Figure 7-4 Block of Data plus Tags

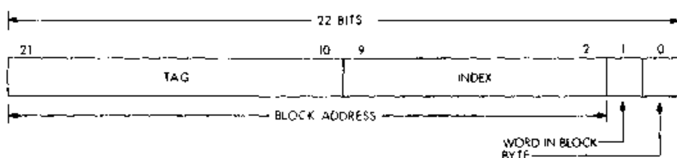


Figure 7-5 Physical Address

The elements of the cache must store not only the data, but also the address identification. Since the index position itself implies part of the address, only the high address field (called tag field) must be stored. The combination of the tag plus index gives the address of the two-word block in main memory. The lowest two bits in the physical address select the particular word in the block, and the byte (if needed).

There are two places in the cache where any block of data can go, a particular index position in either Group 0 or Group 1. Random selection determines into which group the information is placed, overwriting the previous data. Another bit is needed within the cache to determine if the block has been loaded with data. When power is first applied, the cache data is invalid, and the valid bit for each data block is cleared. When a particular block location is updated, the associated valid bit is set to indicate good data.

Figure 7-4 shows the organization for a single block of data within a set. Note that data has byte parity, and that the non-data part called tags contains a 12-bit high order address field plus a valid bit and two parity bits.

7.2.3 General Operation

The system always looks for data in the fast cache memory first. If it is there (a hit), execution proceeds at the fastest rate. If the information is not there (a miss), and the operation was a read, a two-word block of data is transferred from main memory to the cache. If there is a miss while trying to write, main memory is updated, but there are no changes to the cache. Main memory and the cache are both updated on write hits.

The operation of what happens on hits or misses is summarized in Table 7-1.

Table 7-1 Operation on Hit or Miss

	What Happens In	
	CACHE	MAIN MEMORY
READ		
hit	no change	no change
miss	updated	no change
WRITE		
hit	updated	updated
miss	no change	updated

When power is first applied (Power-Up), all of the valid bits are cleared. If power is suddenly lost, cache data may become invalid, but main memory, with non-volatile core, will have a correct copy of all the data.

With a typical program, writes occur only 10% of the time as compared to 90% of the time for reads. Read hits will average 80 to 95% of all cycles with a typical program.

7.3 PARITY

System Reliability
Parity is used extensively in the PDP-11/70 to ensure the integrity of data storage and transfer, and to enhance the reliability of system operation. All of memory (cache and core) has byte parity. Parity is generated and checked on all transfers between core and cache, again between cache and the CPU, between 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 1 parity bit for each 8-bit byte, refer to Figure 7-6. The cache also stores byte parity for data, and in addition it stores 2 parity bits for the address and control information (tag storage) associated with each 2-word block of data.

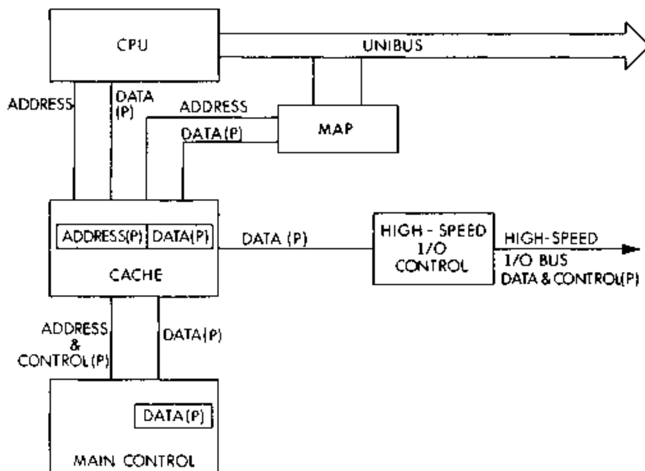


Figure 7-6 Parity (P) in the PDP-11/70 System

The bus between main memory and the cache contains parity on the data and 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.

System Handling of Parity Errors
Extensive capabilities have been designed into the PDP-11/70 to allow recovery from parity errors, and to allow 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 built-in 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, the Memory Management unit 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.

Details of how to perform this programming is explained in the next section on the CPU and memory control registers.

Aborts and Traps
Two actions can take place after detection of a parity error. The cycle can be aborted. Then the computer transfers control through the vector at location 114 to an error handling routine. The other action is that 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; whereas, 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. Refer to Table 7-2.

TABLE 7-2 Response to Parity Errors

PARITY ERROR DETECTED	CONDITION FOR ABORT	CONDITION FOR TRAP
CPU cycle, data error, read from main memory	Error in requested word.	Error in the other word.
UNIBUS cycle,* data error, read from main memory		Error in either word.
CPU cycle, address error, reference to main mem	All reads and writes.	
UNIBUS cycle, address error, reference to main mem		All reads and writes.
CPU or UNIBUS cycle, data or address error, reference to cache		All reads.
High-speed I/O cycle, data or address error, ref to main memory	(no CPU aborts or traps occur; high-speed I/O controllers handle their parity errors).	

* **NOTE:** When a parity error is detected on data going to the UNIBUS, the parity error signal is asserted.

System Response to Parity Errors
Data is read from main memory to the cache in 2-word blocks. If the read cycle was caused by the CPU, and a parity error is detected in the requested word, an abort occurs. If it was in the other word, a trap occurs. On UNIBUS cycles, a trap is caused if there is a read error in either word.

When an address parity error is detected on any read or write to main memory, an abort is caused for both CPU and UNIBUS cycles.

When any fast data memory or address memory parity error is detected on any read from the cache, a trap occurs. On a fast data memory parity error, the CPU will try to get the data from main memory, and also overwrite the same cache location with the new (correct) word just fetched. On an address memory parity error, the CPU will go to main memory for the data, and will correct (overwrite) the tag storage in the cache.

Data transfers for the high-speed mass storage devices take place with main memory. No data is stored in the cache. Parity errors are handled by the device controllers; no CPU aborts or traps occur, and no cache status registers are affected.

Table 7-2 summarizes the system response.

7.4 REGISTERS
The registers described in this section provide information about parity errors, memory status, and CPU status. These hardware registers have program addresses in the top 4K words of physical address space (Peripheral Page).

Register	Address
Low Error Address	17 777 740
High Error Address	17 777 742
Memory System Error	17 777 744
Control	17 777 746
Maintenance	17 777 750
Hit/Miss	17 777 752

Some bit positions of the registers are not used (not implemented with hardware) and are indicated by cross-hatching. These bits are always read as ZEROS by the program. Most of the bits can be read or written under program control.

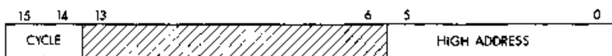
Low Error Address Register 17 777 740

15 — 1	0
LOW ADDRESS (16 BITS)	LSB

This register contains the lowest 16 bits of the 22-bit address of the first error. The least significant bit is bit 0. The high order bits are contained in the High Error Address Register.

All the bits are read only. The bits are undetermined after a Power Up. They are not affected by a Console Start or RESET instruction.

High Error Address Register 17 777 742



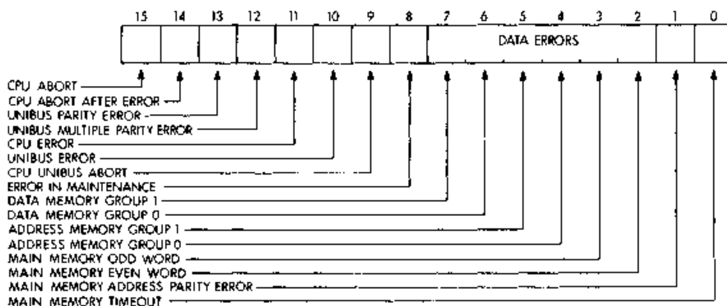
BIT	NAME	FUNCTION
15-14	Cycle Type	These bits are used to encode the type of memory cycle which was being requested when the parity error occurred.

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

BIT	NAME	FUNCTION
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.

All the bits are read only. The bits are undetermined after a Power Up. They are not affected by a Console Start or RESET instruction.

Memory System Error Register 17 777 744



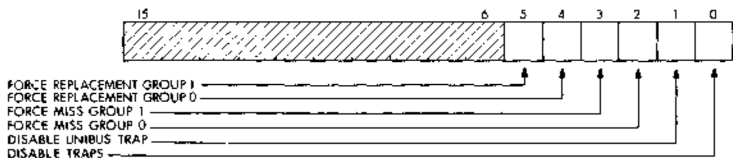
BIT	NAME	FUNCTION
15	CPU Abort	Set if an error occurs which caused the cache to abort a processor cycle.
14	CPU Abort After Error	Set if an abort occurs with the Error Address Register locked by a previous error.
13	UNIBUS Parity Error	Set if an error occurs which resulted in the UNIBUS Map asserting the parity error signal on the UNIBUS.
12	UNIBUS Multiple Parity Error	Set if an error occurs which caused the parity error to be asserted on the UNIBUS with the Error Address Register locked by a previous error.

BIT	NAME	FUNCTION
11	CPU Error	Set if any memory error occurs during a cache CPU cycle.
10	UNIBUS Error	Set if any memory errors occurs during a cache cycle from the UNIBUS.
9	CPU UNIBUS Abort	Set if the processor traps to vector 114 because of UNIBUS parity error on a DATI or DATIP memory cycle.
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. (Main memory always transfers two words at a time.) 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 Memory 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 Timeout	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 set, cause a time-out on the UNIBUS, and then cause the CPU to trap to vector 114.

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 that bit, and it is cleared if a 1 is written to that bit. 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 inadvertantly cleared.

Control Register 17 777 746



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 non-fatal errors.

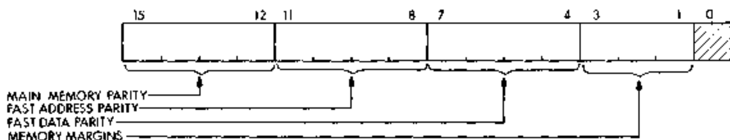
Bits 5 through 0 are read/write. The bits are cleared on Power Up or by Console Start.

The PDP-11/70 has the capability of running in a degraded mode if problems are detected in the cache. If Group 0 of the cache is malfunctioning, it is possible to force all operations through Group 1. If bits 2 and 5 of the Control Register are set, and bits 3 and 4 are clear, 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. But system throughout will not decrease by 50%, since the statistics of read hit probability will still provide reasonably fast operation.

If Group 1 is malfunctioning, bits 3 and 4 should be set, and bits 2 and 5 cleared; such that only Group 0 is operating. If all of the cache is malfunctioning, bits 2 and 3 should be set. The cache will be bypassed, and all references will be to main memory.

Bits 1 and 0 can be set to disable trapping; more memory cycles will be performed, but overall system operation will produce correct results.

Maintenance Register 17 777 750



BIT	NAME	FUNCTION
15-12	Main Memory Parity	Setting these bits causes the 4 parity bits to be 1's. There is 1 bit per byte; there are 4 bytes in the data block. **Bit Set** **Byte** 15 odd word, high byte 14 odd word, low byte 13 even word, high byte 12 even word, low byte
11-8	Fast Address Parity	Setting these bits causes the 4 parity bits for fast address memory to be wrong. Bits 11 and 10 affect Group 1; bits 9 and 8 affect Group 0.
7-4	Fast Data Parity	Setting these bits causes the 4 parity bits to be 1's. **Bit Set** **Byte** 7 Group 1, high byte 6 Group 1, low byte 5 Group 0, high byte 4 Group 0, low byte
3-1	Memory Margins	These bits are encoded to do maintenance checks on main memory.

Bit 3	Bit 2	Bit 1	
0	0	0	Normal operation
0	0	1	Check wrong address parity
0	1	0	Early strobe margin
0	1	1	Late strobe margin
1	0	0	Low current margin
1	0	1	High current margin
1	1	0	(reserved)
1	1	1	(reserved)

All of main memory is margined simultaneously.

Hit/Miss Register 17 777 752

This register indicates whether the 6 most recent references by the CPU were hits or misses. A ONE (1) indicates a read hit; a ZERO (0) 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 Console Start or a RESET instruction.

CHAPTER 8

FLOATING POINT PROCESSOR

8.1 INTRODUCTION

The PDP-11 Floating Point Processor is an optional arithmetic processor which fits into the PDP-11/70 Central Processor. It performs all floating point arithmetic operations and converts data between integer and floating point formats.

The hardware provides a time and money-saving alternative to the use of software floating point routines. Its use can result in many orders of magnitude improvement in the execution of arithmetic operations.

The features of the unit are:

- Overlapped operation with central processor
- High speed
- Single and double precision (32 or 64 bit) floating point modes
- Flexible addressing modes
- Six 64-bit floating point accumulators
- Error recovery aids

8.2 OPERATION

The Floating Point Processor is an integral part of the Central Processor. It operates using similar address modes, and the same memory management facilities provided by the Memory Management Option, as the Central Processor. Floating Point Processor instructions can reference the floating point accumulators, the Central Processor's general registers, or any location in memory.

When, in the course of a program, an FPP Instruction is fetched from memory, the FPP will execute that instruction in parallel with the CPU continuing with its instruction sequence. The CPU is delayed a very short period of time during the FPP Instruction's Fetch operation, and then is free to proceed independently of the FPP. The interaction between the two processors is automatic, and a program can take full advantage of the parallel operation of the two processors by intermixing Floating Point Processor and Central Processor instructions.

Interaction between Floating Point Processor and Central Processor instructions is automatically taken care of by the hardware. When an FPP Instruction is encountered in a program, the machine first initiates Floating Point handshaking and calculates the address of the operand. It then checks the status of the Floating Point Processor. If the FPP is "busy", the CPU will wait until it is "done" before continuing execution of the program. As an example, consider the following sequence of instructions:

```
        LDD(R3)+,AC3     ;Pick up constant operand and place it
                         in AC3
ADDLP:  LDD(R3)+,AC0     ;Load AC0 with next value in table
        MUL AC3,AC0      ;and multiply by constant in AC3
```

```
ADDD AC0,AC1        ;and add the result into AC1
SOB R5,ADDLP        ;check to see whether done
STCDI AC1@R4        ;done, convert double to integer and
                    store
```

In the above example, the Floating Point Processor will execute the first three instructions. After the "ADDD" is fetched into the FPP, the CPU will execute the "SOB", calculate the effective address of the STCDI instruction, and then wait for the FPP to be "done" with the "ADDD" before continuing past the STCDI instruction.

As can be seen from this example, autoincrement and autodecrement addressing automatically adds or subtracts the correct amount to the contents of the register, depending on the modes represented by the instruction.

8.3 ARCHITECTURE

The Floating Point Processor contains scratch registers, a Floating Exception Address pointer (FEA), a Program Counter, a set of Status and Error Registers, and six general purpose accumulators (AC0-AC5).

Each accumulator is interpreted to be 32 or 64 bits long depending on the instruction and the status of the Floating Point Processor. For 32-bit instruction only the left-most 32 bits are used, while the remaining 32 bits remain unaffected.

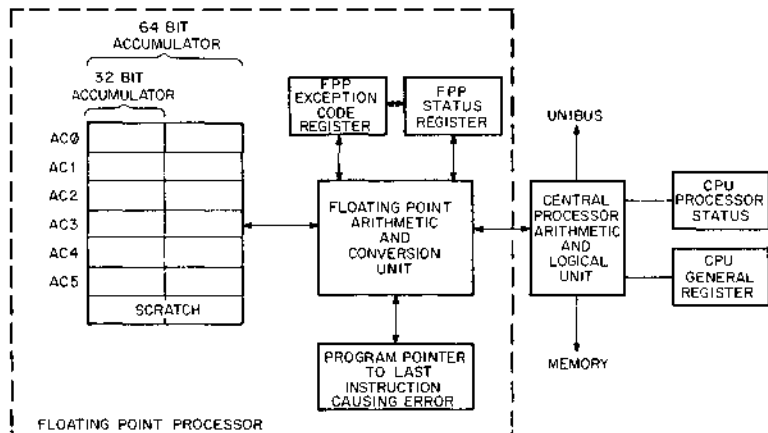


Figure 8.1: Floating Point Processor

The six Floating Point Accumulators are used in numeric calculations and interaccumulator data transfers; the first four (AC0-AC3) are also used for all data transfers between the FPP and the General Registers or Memory.

8.4 FLOATING POINT DATA FORMATS

Mathematically, a floating point number may be defined as having the form (2**K)*f, where K is an integer and f is a fraction. For a non-vanishing number, K and f are uniquely determined by imposing the condition $\frac{1}{2} \leq f < 1$. The fractional part, f, of the number is then said to be normalized. For the number zero, f must be assigned the value 0, and the value of K is indeterminate.

The FPP floating point data formats are derived from this mathematical representation for floating point numbers. Two types of floating point data are provided. In single precision, or Floating Mode, the word is 32 bits long. In double precision, or Double Mode, the word is 64 bits long. Sign magnitude notation is used.

8.4.1. Non-vanishing Floating Point Numbers

The fractional part f is assumed normalized, so that its most significant bit must be 1. This 1 is the "hidden" bit: it is not stored in the data word, but of course the hardware restores it before carrying out arithmetic operations. The Floating and Double modes reserve 23 and 55 bits, respectively, for f, which with the hidden bit, imply effective word lengths of 24 bits and 56 bits for arithmetic operations.

Eight bits are reserved for the storage of the exponent K in excess 128 (200 octal) notation (i.e. as K + 200 octal). Thus exponents from −128 to +127 could be represented by 0 to 377 (octal), or 0 to 255 (decimal). For reasons given below, a biased EXP of 0 (true exponent of −200 octal), is reserved for floating point zero. Thus exponents are restricted to the range −127 to +127 inclusive (−177 to 177 octal) or, in excess 200 (octal) notation, 1 to 377 (octal).

The remaining bit of the floating point word is the sign bit.

8.4.2. Floating Point Zero

Because of the hidden bit, the fractional part is not available to distinguish between zero and non-vanishing numbers whose fractional part is exactly 1/2. Therefore the FP11 reserves a biased exponent of 0 for this purpose. And any floating point number with biased exponent of 0 either traps or is treated as if it were an exact 0 in arithmetic operations. An exact zero is represented by a word, whose bits are all 0's. An arithmetic operation for which the resulting true exponent exceeds 177 (octal) is regarded as producing a floating overflow; if the true exponent is less than −177 (octal) the operation is regarded as producing a floating underflow. A biased exponent of 0 can thus arise from arithmetic operations as a special case of overflow (true exponent = 400 octal), or as a special case of underflow (true exponent = 0). (Recall that only eight bits are reserved for the biased exponent.) The fractional part of results obtained from such overflows and underflows is correct.

8.4.3. The Undefined Variable

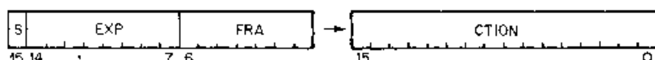
The undefined variable is defined to be any bit pattern with a sign bit of one and a biased exponent of zero. The term "undefined variable" is used, for historical reasons, to indicate that these bit patterns are not assigned a corresponding floating point arithmetic value. Note that the undefined variable is frequently referred to as "−0" elsewhere in this chapter.

A design obective of the FP11C was to assure that the undefined variable would not be stored as the result of any floating point operation in a program run with the overflow and underflow interrupts disabled. This is achieved by storing an exact zero on overflow or underflow, if the corresponding interrupt is disabled. This feature together with an ability to detect a reference to the undefined variable (implemented by the FIUV bit discussed in the next section) is intended to provide the user with a debugging aid: if the presence −0 occurs, it did not result from a previous floating point arithmetic instruction.

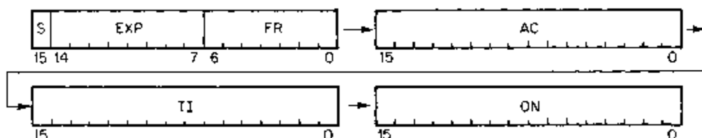
8.4.4. Floating Point Data

Floating point data is stored in words of memory as illustrated below.

F Format, single precision



D Format, double precision



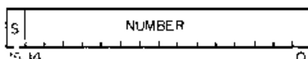
S = Sign of Fraction

EXP = Exponent in excess 200 notation, restricted to 1 to 377 octal for non-vanishing numbers.

FRACTION = 23 bits in F Format, 55 bits in D Format, + one hidden bit (normalization). The binary radix point is to the left.

The FPP provides for conversion of Floating Point to Integer Format and vice-versa. The processor recognizes single precision integer (I) and double precision integer long (L) numbers, which are stored in standard two's complement form:

I Format:



L Format:

where

S = Sign of Number

NUMBER = 15 bits in I Format, 31 bits in L Format.

8.5 FLOATING POINT UNIT STATUS REGISTER (FPS register)

This register provides (1) mode and interrupt control for the floating point unit, and (2) conditions resulting from the execution of the previous instruction.

Four bits of the FPS register control the modes of operation:

> Single/Double: Floating point numbers can be either single or double precision.
>
> Long/Short: Integer numbers can be 16 bits or 32 bits.
>
> Chop/Round: The result of a floating point operation can be either chopped or rounded. The term "chop" is used instead of "truncate" in order to avoid confusion with truncation of series used in approximations for function subroutines.
>
> Normal/Maintenance: a special maintenance mode is available.

The FPS register contains an error flag and four condition codes (5 bits):

> Carry, overflow, zero, and negative, which are equivalent to the CPU condition codes.
>
> The floating point processor (FPP) recognizes seven "floating point exceptions":
>
> > detection of the presence of the undefined variable in memory
> > floating overflow
> > floating underflow
> > failure of floating to integer conversion
> > maintenance trap
> > attempt to divide by zero
> > illegal floating OP code
>
> For the first five of these exceptions, bits in the FPS register are available to individually enable or disable interrupts. An interrupt on the occurrence of either of the last two exceptions can be disabled only by setting a bit which disables interrupts on all seven of the exceptions, as a group.
>
> Of the fourteen bits described above, five are set by the FPP as part of the output of a floating point instruction: the error flag and condition codes. Any of the mode and interrupt control bits (except the FMM bit) may be set by the user; the LDFS instruction is available for this purpose. These fourteen bits are stored in the FPS register as follows:

FER	FID	UNUSED		FIUV	FIU	FIV	FIC	FD	FL	FT	FMM	FN	FZ	FV	FC
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

BIT	NAME	DESCRIPTION
15	Floating Error (FER)	The FER bit is set by the FPP if 1. division by zero occurs 2. illegal OP code occurs 3. any one of the remaining occurs and the corresponding interrupt is enabled. Note that the above action is independent of whether the FID bit (next item) is set or clear. Note also that the FPP never resets the FER bit. Once the FER bit is set by the FPP, it can be cleared only by an LDFPS instruction (or by the RESET instruction described in Section 4.7). This means that the FER bit is up to date only if the most recent floating point instruction produced a floating point exception.
14	Interrupt Disable (FID)	If the FID bit is set, all floating point interrupts are disabled. Note that if an individual interrupt is simultaneously enabled, only the interrupt is inhibited; all other actions associated with the individual interrupt enabled take place.

NOTES

1. The FID bit is primarily a maintenance feature. It should normally be clear. In particular, it must be clear if one wishes to assure that storage of —0 by the FP11C is always accompanied by an interrupt.
2. Through the rest of this chapter, it is assumed that the FID bit is clear in all discussions involving overflow, underflow, occurrence of —0, and integer conversion errors.

BIT	NAME	DESCRIPTION
13	Not Used	
12	Not used	
11	Interrupt on Undefined Variable (FIUV)	An interrupt occurs if FIUV is set and a —0 is obtained from memory as an operand of ADD, SUB, MUL, DIV, CMP, MOD, NEG, ABS, TST or any LOAD instruction. The interrupt occurs before execution on the FP11B. It also occurs before execution

BIT	NAME	DESCRIPTION
		on the FP11C except on NEG and ABS for which it occurs after execution. When FIUV is reset, −0 can be loaded and used in any FPP operation. Note that the interrupt is not activated by the presence of −0 in an AC operand of an arithmetic instruction: in particular, trap on −0 never occurs in Mode 0. The FP11C will not store a result of −0 without the simultaneous occurrence of an interrupt (See Section 8.4).
10	Interrupt on Underflow (FIU)	When the FIU bit is set, Floating Underflow will cause an interrupt. The fractional part of the result of the operation causing the interrupt will be correct. The biased exponent will be too large by 400 (octal), except for the special case of 0, which is correct. An exception is discussed in the detailed description of the LDEXP instruction. If the FIU bit is reset and if underflow occurs, no interrupt occurs and the result is set to exact 0.
9	Interrupt on Overflow (FIV)	When the FIV bit is set, Floating Overflow will cause an interrupt. The fractional part of the result of the operation causing the overflow will be correct. The biased exponent will be too small by 400 (octal). If the FIV bit is reset, and overflow occurs, there is no interrupt. The FP11C returns exact 0; the FP11B returns the result of the operation, just as for FIV set. Special cases of overflow are discussed in the detailed descriptions of the MOD and LDEXP instructions.
8	Interrupt on Integer Conversion Error (FIC)	When the FIC bit is set, and a conversion to integer instruction fails, an interrupt will occur. If

BIT	NAME	DESCRIPTION
		the interrupt occurs, the destination is set to 0, and all other registers are left untouched. If the FIC bit is reset, the result of the operation will be the same as detailed above, but no interrupt will occurr. The conversion instruction fails if it generates an integer with more bits than can fit in the short or long integer word specified by the FL bit (see 6 below).
7	Floating Double Precision Mode (FD)	Determines the precision that is used for floating point calculations. When set, double precision is assumed; when reset, single precision is used.
6	Floating Long Integer Mode (FL)	Active in conversion between integer and floating point format. When set, the integer format assumed is double precision two's complement (i.e. 32 bits). When reset, the integer format is assumed to be single precision two's complement (i.e. 16 bits).
5	Floating Chop Mode (FT)	When bit FT is set, the result of any arithmetic operation is chopped (or truncated). When reset, the result is rounded. See Section 8.8 for a discussion of the chopping and rounding operations.
4	Floating Maintenance Mode (FMM)	This code is a maintenance feature. Refer to the Maintenance Manual for the details of its operation. The FMM bit can be set only in Kernel Mode.
3	Floating Negative (FN)	FN is set if the result of the last operation was negative, otherwise it is reset.
2	Floating Zero (FZ)	FZ is set if the result of the last operation was zero; otherwise it is reset.
1	Floating Overflow (FV)	FV is set if the last operation resulted in an exponent overflow; otherwise it is reset.

BIT	NAME	DESCRIPTION
0	Floating Carry (FC)	FC is set if the last operation resulted in a carry of the most significant bit. This can only occur in floating or double to integer conversions.

8.6 FLOATING EXCEPTION CODE AND ADDRESS REGISTERS

One interrupt vector is assigned to take care of all floating point exceptions (location 244). The seven possible errors are coded in the four bit FEC (Floating Exception Code) register as follows:

2	Floating OP code error
4	Floating divide by zero
6	Floating (or double) to integer conversion error
8	Floating overflow
10	Floating underflow
12	Floating undefined variable
14	Maintenance trap

The address of the instruction producing the exception is stored in the FEA (Floating Exception Address) register.

The FEC and FEA registers are updated only when one of the following occurs:

1. divide by zero
2. illegal OP code
3. any of the other five exceptions with the corresponding interrupt is enabled.

NOTE

1. If one of the last five exceptions occurs with the corresponding interrupt disabled, the FEC and FEA are not updated.
2. Inhibition of interrupts by the FID bit does not inhibit updating of the FEC and FEA, if an exception occurs.
3. The FEC and FEA do not get updated if no exception occurs. This means that the STST (store status) instruction will return current information only if the most recent floating point instruction produced an exception.
4. Unlike the FPS register, no instructions are provided for storage into the FEC and FEA registers.

8.7 FLOATING POINT PROCESSOR INSTRUCTION ADDRESSING

Floating Point Processor instructions use the same type of addressing as the Central Processor instructions. A source or destination operand is specified by designating one of eight addressing modes and one of eight central processor general registers to be used in the specified mode. The modes of addressing are the same as those of the central processor except for mode 0. In mode 0 the operand is located in the designated Floating Point Processor Accumulator, rather than in a Central processor general register. The modes of addressing:

0 = Direct Accumulator

1 = Deferred

2 = Auto-increment

3 = Auto-increment deferred

4 = Auto-decrement

5 = Auto-decrement deferred

6 = Indexed

7 = Indexed deferred

Autoincrement and autodecrement operate on increments and decrements of 4 for F Format and 10_8 for D Format.

In mode 0, the user can make use of all six FPP accumulators (AC0—AC5) as his source or destination. In all other modes, which involve transfer of data from memory or the general register, the user is restricted to the first four FPP accumulators (AC0—AC3).

In immediate addressing (Mode 2, R7) only 16 bits are loaded or stored.

8.8 ACCURACY

General comments on the accuracy of the FPP are presented here. The descriptions of the individual instructions include the accuracy at which they operate. An instruction or operation is regarded as "exact" if the result is identical to an infinite precision calculation involving the same operands. The a priori accuracy of the operands is thus ignored. All arithmetic instructions treat an operand whose biased exponent is 0 as an exact 0 (unless FIUV is enabled and the operand is —0, in which case an interrupt occurs). For all arithmetic operations, except DIV, a zero operand implies that the instruction is exact. The same statement holds for DIV if the zero operand is the dividend. But if it is the divisor, division is undefined and an interrupt occurs.

For non-vanishing floating point operands, the fractional part is binary normalized. It contains 24 bits or 56 bits for Floating Mode and Double Mode, respectively. The internal hardware registers contain 60 bits for processing the fractional parts of the operands, of which the high order bit is reserved for arithmetic overflow. Therefore there are, internally, 35 guard bits for Floating Mode and 3 guard bits for Double Mode arithmetic operations. For ADD, SUB, MUL, and DIV, two guard bits are necessary and sufficient to guarantee return of a chopped or rounded result identical to the corresponding infinite precision operation chopped or rounded to the specified word length. Thus, with two guard bits, a chopped result has an error bound of one least significant bit (LSB); a rounded result has an error bound of 1/2 LSB. (For a radix other than 2, replace "bit" with "digit" in the two preceding sentences to get the corresponding statements on accuracy.) These error bounds are realized by both the FP11B and FP11C for most instructions. For the addition of operands of opposite sign or for the subtraction of operands of the same sign in rounded double precision, the error bound is 9/16 LSB, which is slightly larger than the 1/2 LSB error bound for all other rounded operations.

In the rest of this chapter an arithmetic result is called exact if no non-vanishing bits would be lost by chopping. The first bit lost in chopping

is referred to as the "rounding"bit. The value of a rounded result is related to the chopped result as follows:

1. if the rounding bit is one, the rounded result is the chopped result incremented by an LSB (least significant bit).
2. if the rounding bit is zero, the rounded and chopped results are identical.

It follows that

1. If the result is exact
   rounded value = chopped value = exact value
2. If the result is not exact, its magnitude
   (a) is always decreased by chopping
   (b) is decreased by rounding if the rounding bit is zero
   (c) is increased by rounding if the rounding bit is one.

Occurrence of floating point overflow and underflow is an error condition: the result of the calculation cannot be correctly stored because the exponent is too big to fit into the 8 bits reserved for it. However, the internal hardware has produced the correct answer. For the case of underflow replacement of the correct answer by zero is a reasonable resolution of the problem for many applications. This is done on both the FP11B and FP11C if the underflow interrupt is disabled. The error incurred by this action is an absolute rather than a relative error; it is bounded (in absolute value) by 2**(−128). There is no such simple resolution for the case of overflow. The action taken, if the overflow interrupt is disabled, is described under FIV (bit 9) of Section 8.5.

The FIV and FIU bits (of the floating point status word) provide the user with an opportunity to implement his own fix up of an overflow or underflow condition. If such a condition occurs and the corresponding interrupt is enabled, the hardware stores the fractional part and the low eight bits of the biased exponent. The interrupt will take place and the user can identify the cause by examination of the FV (floating overflow) bit or the FEC (floating exception) register. The reader can readily verify that (for the standard arithmetic operations ADD, SUB, MUL, and DIV) the biased exponent returned by the hardware bears the following relation to the correct exponent generated by the hardware:

1. on overflow: it is too small by 400 octal
2. on underflow: if the biased exponent is 0 it is correct. If it is not 0, it is too large by 400 octal.

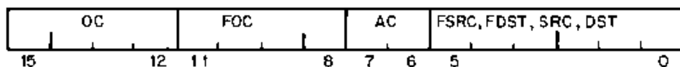
Thus, with the interrupt enabled, enough information is available to determine the correct answer. The user may, for example, rescale his variables (via STEXP and LDEXP) to continue his calculation. Note that the accuracy of the fractional part is unaffected by the occurrence of underflow or overflow.

8.9 FLOATING POINT INSTRUCTIONS

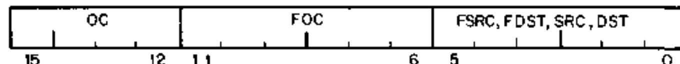
Each instruction that references a floating point number can operate on either floating or double precision numbers depending on the state of the FD mode bit. Similarly, there is a mode bit FL that determines whether a 32-bit integer (FL = 1) or a 16-bit integer (FL = 0)is used in conversion between integer and floating point representation. FSRC and FDST use floating point addressing modes; SRC and DST use CPU addressing Modes.

In the detailed descriptions of the floating point instructions, the operations of the FP11B and FP11C are identical, except where explicitly stated to the contrary.

Floating Point Instruction Format
Double Operand Adressing



Single Operand Addressing

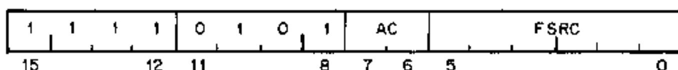


OC = Op Code = 17
FOC = Floating Op Code
AC = Accumulator
FSRC, FDST use FPP Address Modes
SRC, DST use CPU Address Modes

General Definitions:
XL = largest fraction that can be represented:
    1—2**(—24), FD = 0; single precision
    1—2**(—56), FD = 1); double precision
XLL = smallest number that is not identically zero = 2**(—128) —2**(—127))*(1/2)
XUL = largest number that can be represented = 2**(127)*XL
JL = largest integer that can be represented:
    2**(15)—1 if FL = 0   2**(31)—1 if FL = 1
ABS (address) = absolute value of (address)
EXP (address) = biased exponent of (address)
.LT. = "less than"
.LE. = "less than or equal"
.GT. = "greater than"
.GE. = "greater than or equal"
LSB = least significant bit

LDF
LDD

Load Floating/Double 172(AC + 4)FSRC



Operation: AC ← (FSRC)

Condition Codes: FC ← 0
FV ← 0
FZ ← 1 if (AC) = 0, else FZ ← 0.
FN ← 1 if (AC) < 0, else FN ← 0.

Description: Load Single or Double Precision Number into Accumulator.

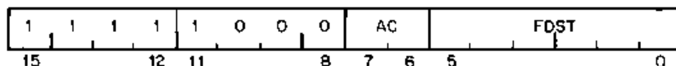
Interrupts: If FIUV is enabled, trap on −0 occurs before AC is loaded. Neither overflow nor underflow can occur.

Accuracy: These instructions are exact.

Special Comment: These instructions permit use of −0 in a subsequent floating point instruction if FIUV is not enabled and (FSRC) = −0.

STF
STD

Store Floating/Double 174ACFDST



Operation: FDST ← (AC)

Condition Codes: FC ← FC
FV ← FV
FZ ← FZ
FN ← FN

Description: Store Single or Double Precision Number from Accumulator.

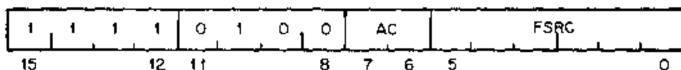
Interrupts: These instructions do not interrupt if FIUV enabled, because the —0, if present, is in AC, not in memory. Neither overflow nor underflow can occur.

Accuracy: These instructions are exact.

Special Comment: These instructions permit storage of a —0 in memory from AC. Note, however, that the FP11C processor can store a —0 in an AC only if it occurs in conjunction with overflow or underflow, and if the corresponding interrupt is enabled. Thus, the user has an opportunity to clear the —0, if he wishes.

ADDF
ADDD

Add Floating/Double 172ACFSRC



Operation: Let SUM = (AC) + (FSRC):

If underflow occurs and FIU is not enabled, AC ← exact 0.

If overflow occurs and FIV is not enabled, AC ← exact 0 on FP11C.

For all other cases, AC ← SUM.

Condition Codes: FC ← 0.
FV ← 1 If overflow occurs, else FV ← 0.
FZ ← 1 If (AC) = 0, else FZ ← 0.
FN ← 1 If (AC) $<$ 0, else FN ← 0.

Description: Add the contents of FSRC to the contents of AC. The addition is carried out in single or double precision and is rounded or chopped in accordance with the values of the FD and FT bits in the FPS register. The result is stored in AC except for:

Overflow with interrupt disabled on the FP11C.

Underflow with interrupt disabled.

For these exceptional cases, an exact 0 is stored in AC.

Interrupts: If FIUV is enabled, trap on —0 in FSRC occurs before execution.

If overflow or underflow occurs and if the corresponding interrupt is enabled, the trap occurs with the faulty result in AC. The fractional parts are correctly stored. The exponent part is too large by 400 octal for underflow, except for the special case of 0, which is correct.

Accuracy: Errors due to overflow and underflow are described above. If neither occurs, then: For oppositely signed operands with exponent differences of 0 or 1, the answer returned is exact if a loss of significance of one or more bits occurs. Note that these are the only cases for which loss of significance of more than one bit can occur. For all other cases the result is inexact with error bounds of

1 LSB in chopping mode with either single or double precision.

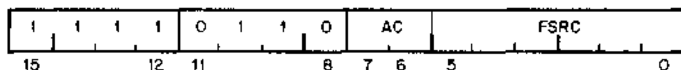
1/2 LSB in rounding mode with single precision.

9/16 LSB in rounding mode with double precision.

Special Comment: The undefined variable −0 can occur only in conjunction with overflow or underflow. It will be stored in AC only if the corresponding interrupt is enabled or, for the FP11B, on overflow even if the overflow interrupt is not enabled.

SUBF
SUBD

Subtract Floating/Double 173ACFSRC



Operation: Let DIFF = (AC) − (FSRC):

If underflow occurs and FIU is not enabled, AC ← exact 0.

If overflow occurs and FIV is not enabled, AC ← exact 0 on the FP11C.

For all other cases, AC ← DIFF.

Condition Codes: FC ← 0.
FV ← 1 If overflow occurs, else FV ← 0.
FZ ← 1 If (AC) = 0, else FZ ← 0.
FN ← 1 If (AC) < 0, else FN ← 0.

Description: Subtract the contents of FSRC from the contents of AC. The subtraction is carried out in single or double precision and is rounded or chopped in accordance with the values of the FD and FT bits in the FPS register. The result is stored in AC except for:

Overflow with interrupt disabled on the FP11C.

Underflow with interrupt disabled.

For these exceptional cases, an exact 0 is stored in AC.

Interrupts: If FIUV is enabled, trap on −0 in FSRC occurs before execution.

If overflow or underflow occurs and if the corresponding interrupt is enabled, the trap occurs with the faulty results in AC. The fractional parts are correctly stored. The exponent part is too small by 400 octal for overflow. It is too large by 400 octal for underflow, except for the special case of 0, which is correct.

Accuracy: Errors due to overflow and underflow are described above. If neither occurs, then: For like-signed operands with exponent difference of 0 or 1, the answer returned is exact if a loss of significance of more than one bit can occur. Note that these are the only cases for which loss of significance of more than one bit can occur. For all other cases the result is inexact with error bounds of

1 LSB in chopping mode with either single or double precision.

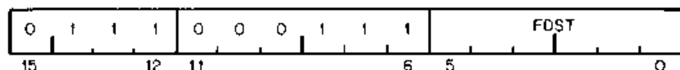
1/2 LSB in rounding mode with single precision.

9/16 LSB in rounding mode with double precision.

Special Comment: The undefined variable —0 can occur only in conjunction with overflow or underflow. It will be stored in the AC only if the corresponding interrupt is enabled or, for the FP11B, on overflow even if the overflow interrupt is not enabled.

NEGF
NEGD

Negate Floating/Double 1707FDST



Operation: FDST ← −(FDST) if EXP(FDST) ≠ 0, else FDST ← exact 0.

Condition Codes:
FC ← 0.
FV ← 0.
FZ ← 1 If EXP(FDST) = 0, else FZ ← 0.
FN ← 1 If (FDST) < 0, else FN ← 0.

Description: Negate single or double Precision number, store result in same location. (FDST)

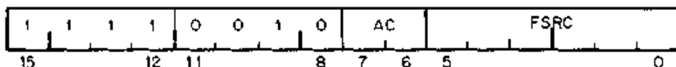
Interrupts: If FIUV is enabled
FP11C: Trap on −0 occurs after execution.
FP11B: Trap on −0 occurs before execution.

Neither overflow nor underflow can occur.

Accuracy: These instructions are exact.

MULF
MULD

Multiply Floating/Double 171ACFSRC



Operation:

Let PROD = (AC)*(FSRC)

If underflow occurs and FIU is not enabled, AC ← exact 0.

If overflow occurs and FIV is not enabled, AC ← exact 0 on FP11C.

For all other cases AC ←PROD

Condition Codes:

FC ← 0.
FV ← 1 if overflow occurs, else FV ← 0.
FZ ← 1 if (AC) = 0, else FZ ← 0.
FN ← 1 if (AC) < 0, else FN ← 0.

Description:

If the biased exponent of either operand is zero, (AC) ← exact 0. For all other cases PROD is generated to 48 bits for Floating Mode and 59 bits for Double Mode. The product is rounded or chopped for FT = 0 and 1, respectively, and is stored in AC except for

Overflow with interrupt disabled on the FP11C.

Underflow with interrupt disabled.

For these exceptional cases, an exact 0 is stored in accumulator.

Interrupts:

If FIUV is enabled, trap on −0 occurs before execution.

If overflow or underflow occurs and if the corresponding interrupt is enabled, the trap occurs with the faulty results in AC. The fractional parts are correctly stored. The exponent part is too small by 400 octal for overflow. It is too large by 400 otcal for underflow, except for the special case of 0, which is correct.

Accuracy:

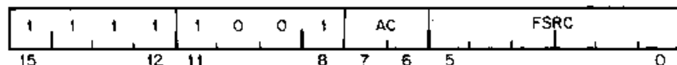
Errors due to overflow and underflow are described above. If neither occurs, the error incurred is bounded by 1 LSB in chopping mode and 1/2 LSB in rounding mode.

Special Comment:

The undefined variable −0 can occur only in conjunction with overflow or underflow. It will be stored in AC only if corresponding interrupt is enabled or, for the FP11B, on overflow even if the overflow interrupt is not enabled.

DIVF
DIVD

Divide Floating/Double 174(AC + 4)FSRC



Operation: If EXP(FSRC) = 0, AC ← (AC): instruction is aborted.

If EXP(AC) = 0, AC ← exact 0.

For all other cases, let QUOT = (AC)/(FSRC):

If underflow occurs and FIU is not enabled AC ← exact 0.

If overflow occurs and FIV is not enabled, AC ← exact 0 on the FP11C.

For all remaining cases AC ← QUOT.

Condition Codes: FC ← 0.
FV ← 1 if overflow occurs, else FV ← 0.
FZ ← 1 if EXP(AC) = 0, else FZ ← 0.
FN ← 1 if (AC) < 0, else FN ← 0.

Description: If either operand has a biased exponent of 0, it is treated as an exact 0. For FSRC this would imply division by zero; in this case the instruction is aborted, the FEC register is set to 4 and an interrupt occurs. Otherwise the quotient is developed to single or double precision with enough guard bits for correct rounding. The quotient is rounded or chopped in accordance with the values of the FD and FT bits in the FPS register. The result is stored in AC except for:

Overflow with interrupt disabled on the FP11C.

Underflow with interrupt disabled.

For these exceptional cases an exact 0 is stored in accumulator.

Interrupts: If FIUV is enabled, trap on −0 in FSRC occurs before execution.

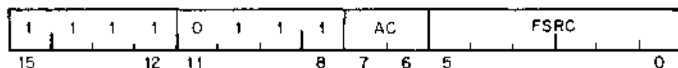
If EXP(FSRC) = 0 interrupt traps on attempt to divide by 0.

If overflow or underflow occurs and if the corresponding interrupt is enabled, the trap occurs with the faulty results in AC. The fractional parts are correctly stored. The exponent part is too small by 400 octal for overflow. It is too large by 400 octal for underflow, except for the special case of 0, which is correct.

Accuracy: Errors due to overflow, underflow and division by 0 are described above. If none of these occurs, the error in the quotient will be bounded by 1 LSB in chopping mode and by 1/2 LSB in rounding mode.

Special Comment: The undefined variable —0 can occur only in conjunction with overflow or underflow. It will be stored in AC only if the corresponding interrupt is enabled or, for the FP11B, on overflow even if the overflow interrupt is not enabled.

Compare Floating/Double 173 (AC + 4) FSRC



Operation: (FSRC) − (AC)

Condition Codes:
FC ← 0.
FV ← 0.
FZ ← 1 If (FSRC) − (AC) = 0, else FZ ← 0.
FN ← 1 If (FSRC) − (AC) < 0, else FN ← 0.

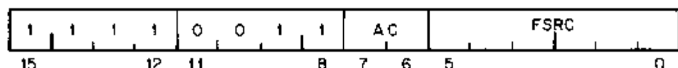
Description: Compare the contents of FSRC with the accumulator. Set the appropriate floating point condition codes. FSRC and the accumulator are left unchanged.

Interrupts: If FIUV is enabled, trap on −0 occurs before execution.

Accuracy: These instructions are exact.

MODF
MODD

Multiply and Integerize Floating/Double 171(AC + 4)FSRC



Description and Operation

This instruction generates the product of its two floating point operands, separates the product into integer and fractional parts and then stores one or both parts as floating point numbers.

Let PROD = (AC)*(FSRC) so that in:

Floating point: ABS(PROD) = (2**K) *f

where 1/2.LE.f.LT.1 and
EXP(PROD) = (200 + K) octal

Fixed Point binary: PROD = N + g, with

N = INT(PROD) = the integer part of PROD

and

g = PROD — INT(PROD) = the fractional part of PROD with 0.LE.g.LT.1

Both N and g have the same sign as PROD. They are returned as follows:

If AC is an even-numbered accumulator (0 or 2), N is stored in AC + 1 (1 or 3), and g is stored in AC.

If AC is an odd-numbered accumulator, N is not stored, and g is stored in AC.

The two statements above can be combined as follows: N is returned to ACv1 and g is returned to AC, where v means .OR.

Five special cases occur, as indicated in the following formal description with L = 24 for Floating Mode and L = 56 for Double Mode:

1. If PROD overflows and FIV enabled:

ACv1 ← N, chopped to L bits, AC ← exact 0

Note that EXP(N) is too small by 400 (octal), and that ←0 can get stored in ACv1.

If FIV is not enabled: action is same as above for FP11B. For FP11C, ACv1 ← exact 0, AC ← exact 0, and —0 will never be stored.

2. If 2**L.LE.ABS(PROD) and no overflow

ACv1 ← N, chopped to L bits, AC ← exact 0

The sign and EXP of N are correct, but low order bit information, such as parity, is lost.

3. If 1.LE.ABS(PROD).LT.2**L

   ACv1 ← N, AC ← g

The integer part N is exact. The fractional part g is normalized, and chopped or rounded in accordance with FT. Rounding may cause a return of ±unity for the fractional part. For L = 24, the error in g is bounded by 1 LSB in chopping mode and by 1/2 LSB in rounding mode. For L = 56, the error in g increases from the above limits as ABS(N) increases above 3 because only 59 bits of PROD are generated:

   if 2**p.LE.ABS(N).LT.2**(p + 1), with p > 2, the low order p − 2 bits of g may be in error.

4. If ABS (PROD). LT.1 and no underflow:

   ACv1 ← exact 0 AC ← g

There is no error in the integer part. The error in the fractional part is bounded by 1 LSB in chopping mode and 1/2 LSB in rounding mode. Rounding may cause a return of ±unity for the fractional part.

5. If PROD underflows and FIU enabled:

   ACv1 ← exact 0 AC ← g

Errors are as in case 4, except that EXP(AC) will be too large by 400 octal (except if EXP = 0, it is correct). Interrupt will occur and −0 can be stored in AC.

IF FIU is not enabled, ACv1 ← exact 0 and AC ← exact 0. For this case the error in the fractional part is less than 2**(−128).

Condition Codes:

FC ← 0.
FV ← 1 if PROD overflows on FP11C, else
   FV ← 0.
FZ ← 1 if (AC) = 0, else FZ ← 0.
FN ← if (AC) < 0, else FN ← 0.

Interrupts:

If FIUV is enabled, trap on −0 in FSRC will occur before execution.

Overflow and Underflow are discussed above.

Accuracy:

Discussed above.

Applications:

1. Binary to decimal conversion of a proper fraction: the following algorithm, using MOD, will generate decimal digits D(1), D(2) . . . from left to right:

Initialize: I ← 0
   X ← number to be converted;
   ABS(X) < 1

```
While X ≠ 0 do
Begin PROD ← X*10;
   I ← I + 1;
   D(I) ← INT(PROD);
   X ← PROD − INT(PROD);
   END;
```

This algorithm is exact; it is case 3 in the description: the number of non-vanishing bits in the fractional part of PROD never exceeds L, and hence neither chopping nor rounding can introduce error.

2. To reduce the argument of a trigonometric function.

ARG*2/PI = N + g. The low two bits of N identify the quadrant, and g is the argument reduced to the first quadrant. The accuracy of N +g is limited to L bits because of the factor 2/PI. The accuracy of the reduced argument thus depends on the size of N.

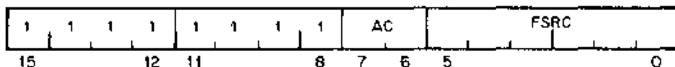
3. To evaluate the exponential function e**x, obtain

   x*(log e base 2) = N + g.
   Then e**x = (2**N)*(e**(g*1n 2))

The reduced argument is $g*1n2 < 1$ and the factor 2**N is an exact power of 2, which may be scaled in at the end via STEXP, ADD N to EXP and LDEXP. The accuracy of N + g is limited to L bits because of the factor (log e base 2). The accuracy of the reduced argument thus depends on the size of N.

LDCDF
LDCFD

Load and convert from Double to Floating or from Floating to Double — 177(AC + 4)FSRC



Operation:

If EXP(FSRC) = 0, AC ← exact 0.

If FD =1, FT = 0, FIV = 0 and rounding causes overflow, AC ← exact 0 on the FP11C.

In all other cases AC ← C_{xy} (FSRC), where C_{xy} specifies conversion from floating mode x to floating y;

x = F, y = D if FD = 0
x = D, y = F if FD = 1.

Condition Codes:

FC ← 0.
FV ← 1 if conversion produces overflow, else FV ← 0.
FZ ← 1 if (AC) = 0, else FZ ← 0.
FN ← 1 if (AC) < 0, else FN ← 0.

Description:

If the current mode is Floating Mode (FD = 0) the source is assumed to be a double-precision number and is converted to single precision. If the Floating Chop bit (FT) is set, the number is chopped, otherwise the number is rounded.

If the current mode is Double Mode (FD = 1), the source is assumed to be a single-precision number, and is loaded left justified in the AC. The lower half of the AC is cleared.

Interrupts:

If FIUV is enabled, trap on —0 occurs before execution.

Overflow cannot occur for LDCFD.

A trap occurs if FIV is enabled, and if rounding with LDCDF causes overflow; AC ← overflowed result of conversion. This result must be +0 or —0.

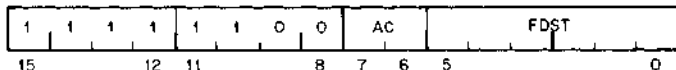
Underflow cannot occur.

Accuracy:

LDCFD is an exact instruction. Except for overflow, described above, LDCDF incurs an error bounded by one LSB in chopping mode, and by 1/2 LSB in rounding mode.

STCFD
STCDF

Store and convert from Floating to Double or from Double to Floating — 176ACFDST



Operation: If EXP(AC) = 0, FDST ← exact 0

If FD = 1, FT = 0, FIV = 0 and rounding causes overflow, FDST ← exact 0 on FP11C.

In all other cases, FDST ← C_{xy}(AC), where C_{xy} specifies conversion from floating mode x to floating mode y;

x = F and y = D if FD = 0,
x = D and y = F if FD = 1.

Condition Codes: FC ← 0.
FV ← 1 If conversion produces overflow else FV ← 0.
FZ ← 1 If (AC) = 0, else FZ ← 0.
FN ← 1 If (AC) < 0, else FN ← 0.

Description: If the current mode is single precision, the Accumulator is stored left justified in FDST and the lower half is cleared. If the current mode is double precision, the contents of the accumulator are converted to single precision, chopped or rounded depending on the state of FT, and stored in FDST.

Interrupts: Trap on −0 will not occur even if FIUV is enabled because FSRC is an accumulator.

Underflow cannot occur.

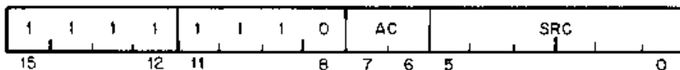
Overflow cannot occur for STCFD.

A trap occurs if FIV is enabled, and if rounding with STCDF causes overflow; FDST ← overflowed result of conversion. This result must be +0 or −0.

Accuracy: STCFD is an exact instruction. Except for overflow, described above, STCDF incurs an error bounded by 1 LSB in chopping mode and 1/2 LSB in rounding mode.

LDCIF
LDCID
LDCLF
LDCLD

Load and Convert Integer or Long Integer to Floating or Double Precision 177ACSRC



Operation: AC ← C_{jx} (SRC), where

C_{jx} specifies conversion from integer mode j to floating mode x;

j = I if FL = 0, j = L if FL = 1,
x = F if FD = 0, x = D if FD = 1.

Condition Codes: FC ← 0.
FV ← 0.
FZ ← 1 if (AC) = 0, else FZ ← 0.
FN ← 1 if (AC) $<$ 0, else FN ← 0.

Description: Conversion is performed on the contents of SRC from a 2's complement integer with precision j to a floating point number of precision x. Note that j and x are determined by the state of the mode bits FL and FD: J = I or L, and X = F or D.

If a 32-bit Integer is specified (L mode) and (SRC) has an addressing mode of 0, or immediate addressing mode is specified, the 16 bits of the source register are left justified and the remaining 16 bits loaded with zeroes before conversion.

In the case of LDCLF the fractional part of the floating point representation is chopped or rounded to 24 bits for FT = 1 and 0 respectively.

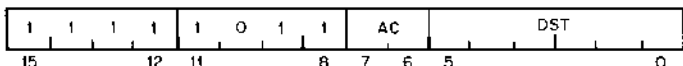
Interrupts: None; SRC is not floating point, so trap on −0 cannot occur.

Overflow and underflow cannot occur.

Accuracy: LDCIF, LDCID, LDCLD are exact instructions. The error incurred by LDCLF is bounded by one LSB in chopping mode, and by 1/2 LSB in rounding mode.

STCFI
STCFL
STCDI
STCDL

Store and Convert from Floating or Double to Integer or Long Integer — 175(AC + 4)DST



Operation: DTS ← C_{xj} (AC) if $-JL-1 < C_{xj}$ (AC) $< JL+1$, else DST ← 0, where C_{xj} specifies conversion from floating mode x to integer mode j;

j = I if FL = 0, j = L if FL = 1,
x = F if FD = 0, x = D if FD = 1.

JL is the largest integer:

2**15 − 1 for FL = 0
2**31 − 1 for FL = 1

Condition Codes: C ← FC ← 0 if $-JL-1 < C_{xj}$ (AC) $< JL+1$, else FC ← 1.
V ← FV ← 0.
Z ← FZ ← 1 if (DST) = 0, else FZ ← 0.
N ← FN ← 1 if (DST) < 0, else FN ← 0.

Description: Conversion is performed from a floating point representation of the data in the accumulator to an integer representation.

If the conversion is to a 32-bit word (L mode) and an address mode of 0, or immediate adressing mode, is specified, only the most significant 16 bits are stored in the destination register.

If the operation is out of the integer range selected by FL, FC is set to 1 and the contents of the DST are set to 0.

Numbers to be converted are always chopped (rather than rounded) before conversion. This is true even when the Chop Mode bit, FT is cleared in the Floating Point Status Register.

Interrupts: These instructions do not interrupt if FIUV is enabled, because the −0, if present, is in AC, not in memory.

If FIC enabled, trap on conversion failure will occur.

Accuracy: These instructions store the integer part of the floating point operand, which may not be the integer most closely approximating the operand. They are exact if the integer part is within the range implied by FL.

LDEXP

Load Exponent 176(AC + 4)SRC

Operation:

NOTE: 177 and 200, appearing below, are octal numbers.

If $-200 < SRC < 200$, EXP(AC) ←(SRC) + 200 and the rest of AC is unchanged on both FP11C and FP11B.

If SRC > 177 and FIV is enabled,
EXP(AC) ← (SRC) < 6:0 > on FP11C,
EXP(AC) ← (SRC) < 7:0 > on FP11B.

If SRC > 177 and FIV is disabled
AC ← exact 0 on FP11C,
EXP(AC) ← (SRC + 200) < 7:0 > on FP11B.

If SRC < −177 and FIU is disabled,
AC ← exact 0 on both FP11C and FP11B.

If SRC < −177 and FIU is enabled,
EXP(AC) ← (SRC) < 6:0 > on FP11C,
EXP(AC) ← (SRC) + 200) < 7:0 > on FP11B.

Condition Codes:

FC ← 0.
FV ← 1 if (SRC) > 177, else FV ← 0.
FZ ← 1 if EXP(AC) = 0, else FZ ← 0.
FN ← 1 if (AC) < 0, else FN ← 0.

Description:

Change AC so that its unbiased exponent = (SRC). That is, convert (SRC) from 2's complement to excess 200 notation, and insert in the EXP field of AC. This is a meaningful operation only if ABS(SRC).LE.177.

If SRC > 177, result is treated as overflow. If SRC < −177, result is treated as underflow. Note that the FP11C and FP11B do not treat these abnormal conditions in exactly the same way.

Interrupts:

No trap on −0 in AC occurs, even if FIUV enabled.

If SRC > 177 and FIV enabled, trap on overflow will occur.

If SRC < −177 and FIU enabled, trap on underflow will occur.

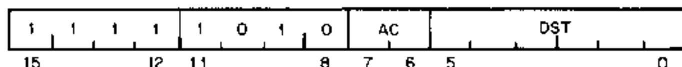
The answers returned by the FP11C and FP11B differ for overflow and underflow conditions.

Accuracy: Errors due to overflow and underflow are described above. If EXP(AC) = 0 and SRC ≠ −200, (AC) changes from a floating point number treated as 0 by all floating arithmetic operations to a non-zero number. This is because the insertion of the "hidden" bit in the hardware implementation of arithmetic instructions is triggered by a non-vanishing value of EXP.

For all other cases, LDEXP implements exactly the transformation of a floating point number (2**K)*f into (2**(SRC))*f where 1/2 .LE.ABS (f).LT.1.

STEXP

Store Exponent 175ACDST



Operation: DST ← EXP(AC)—200 octal

Condition Codes:
C ← FC ← 0.
V ← FV ← 0.
Z ← FZ ← 1 if (DST) = 0, else FZ ← 0.
N ← FN ← 1 if (DST) < 0, else FN ← 0.

Description: Convert accumulator's exponent from excess 200 octal notation to 2's complement, and store result in DST.

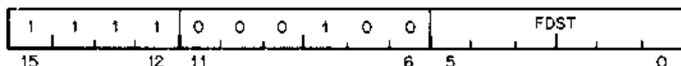
Interrupts: This instruction will not trap on —0.

Overflow and underflow cannot occur.

Accuracy: This instruction is always exact.

CLRF
CLRD

Clear Floating/Double 1704FDST



Operation: FDST ← exact 0.

Condition Codes: FC ← 0.
FV ← 0.
FZ ← 1
FN ← 0.

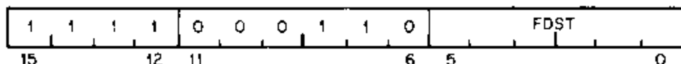
Description: Set FDST to 0. Set FZ condition code and clear other condition code bits.

Interrupts: No interrupts will occur. Neither overflow nor underflow can occur.

Accuracy: These instructions are exact.

ABSF
ABSD

Make Absolute Floating/Double 1706FDST



Operation: If (FDST) < 0, FDST ← −(FDST).

If EXP(FDST) = 0, FDST ← exact 0.

For all other cases, FDST ← (FDST).

Condition Codes: FC ← 0.
FV ← 0.
FZ ← 1 if EXP(FDST) = 0, else FZ ← 0.
FN ← 0

Description: Set the contents of FDST to its absolute value.

Interrupts: If FIUV is set:

FP11C: Trap on −0 occurs after execution

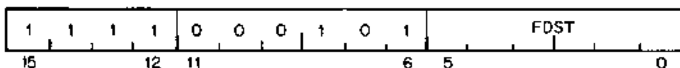
FP11B: Trap on −0 occurs before execution

Overflow and underflow cannot occur.

Accuracy: These instructions are exact.

TSTF
TSTD

Test Floating/Double 1705FDST



Operation: FDST ← (FDST)

Condition Codes:
FC ← 0.
FV ← 0.
FZ ← 1 if EXP(FDST) = 0, else FZ ← 0.
FN ← 1 if (FDST) < 0, else FN ← 0.

Description: Set the Floating Point Processor's Condition Codes according to the contents of FDST.

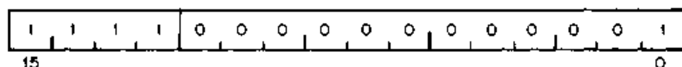
Interrupts: If FIUV is set, trap on −0 occurs after execution

Overflow and underflow cannot occur.

Accuracy: These instructions are exact.

SETF

Set Floating Mode 170001

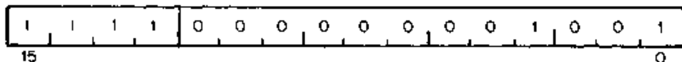


Operation: FD ← 0

Description: Set the FPP in Single Precision Mode.

SETD

Set Floating Double Mode 170011

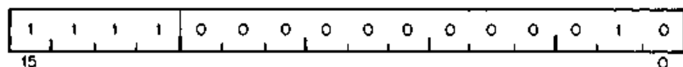


Operation: FD ← 1

Description: Set the FPP in Double Precision Mode.

SETI

Set Integer Mode 170002

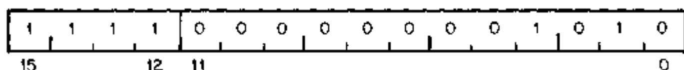


Operation: FL ← 0

Description: Set the FPP for Integer Data.

SETL

Set Long Integer Mode 170012

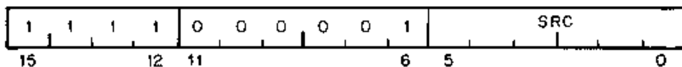


Operation: FL ← 1

Description: Set the FPP for Long Integer Data.

LDFPS

Load FPPs Program Status 1701SRC

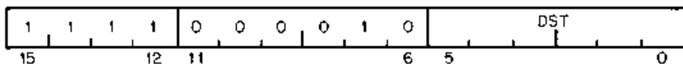


Operation: FPS ← (SRC)

Description: Load FPP's Status from SRC.

STFPS

Store FPPs Program Status 1702DST

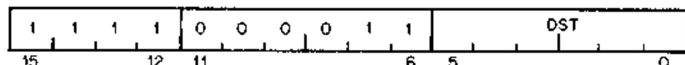


Operation: DST ← (FPS)

Description: Store FPP's Status in DST.

STST

Store FPPs Status 1703DST



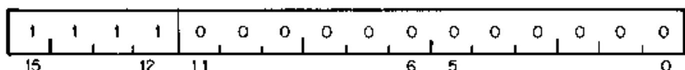
Operation: DST ← (FEC)
DST + 2 ←(FEA)

Description: Store the FEC and then the FPP's Exception Address Pointer in DST and DST + 2.

NOTES:
1. If destination mode specifies a general register or immediate addressing, only the FEC is saved.
2. The information in these registers is current only if the most recently executed floating point instruction (refer to Section 8.6) caused a floating-point exception.

CFCC

Copy Floating Condition Codes 170000



Operation: C ← FC
V ← FV
Z ← FZ
N ← FN

Description: Copy FPP Condition Codes into the CPU's Condition Codes.

CHAPTER 9

PROGRAMMING TECHNIQUES

In order to produce programs which fully utilize the power and flexibility of the PDP-11, the reader should become familiar with the various programming techniques which are part of the basic design philosophy of the PDP-11. Although it is possible to program the PDP-11 along traditional lines such as "accumulator orientation" this approach does not fully exploit the architecture and instruction set of the PDP-11.

9.1 THE STACK

A "stack," as used on the PDP-11, is an area of memory set aside by the programmer for temporary storage or subroutine/interrupt service linkage. The instructions which facilitate "stack" handling are useful features not normally found in low-cost computers. They allow a program to dynamically establish, modify, or delete a stack and items on it. The stack uses the "last-in, first-out" concept; that is, various items may be added to a stack in sequential order and retrieved or deleted from the stack in reverse order. On the PDP-11, a stack starts at the highest location reserved for it and expands linearly downward to the lowest address as items are added to the stack.

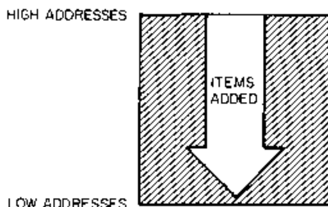


Figure 9-1: Stack Addresses

To keep track of the last item added to the stack (or "where we are" in the stack) a General Register always contains the memory address where the last item is stored in the stack. In the PDP-11 any register except Register 7 (the Program Counter-PC) may be used as a "stack pointer" under program control; however, instructions associated with subroutine linkage and interrupt service automatically use Register 6 (R6) as a hardware "Stack Pointer." For this reason R6 is frequently referred to as the system "SP".

Stacks in the PDP-11 may be maintained in either full word or byte units. This is true for a stack pointed to by any register except R6, which must be organized in full word units only.

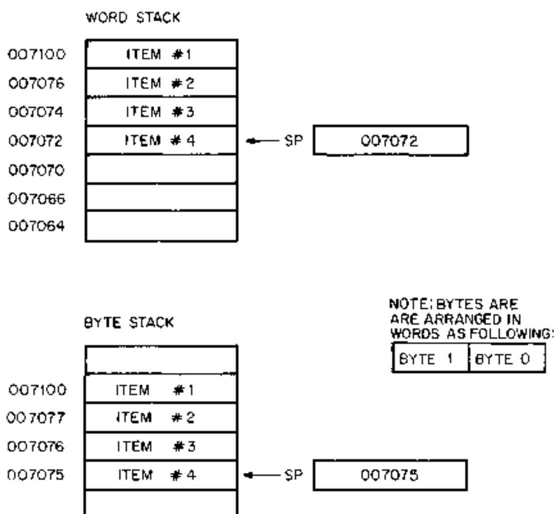


Figure 9-2: Word and Byte Stacks

Items are added to a stack using the autodecrement addressing mode with the appropriate pointer register. (See Chapter 3 for description of the autoincrement/decrement modes).

This operation is accomplished as follows:

```
MOV Source,—(SP)          ;MOV Source Word onto the stack
                            or
MOVB Source,—(SP)         ;MOVB Source Byte onto the stack
```

This is called a "push" because data is "pushed onto the stack."

To remove an item from a stack the autoincrement addressing mode with the appropriate SP is employed. This is accomplished in the following manner:

```
MOV(SP)+,Destination      ;MOV Destination Word off the stack
                            or
MOVB(SP)+,Destination     ;MOVB Destination Byte off the stack
```

Removing an item from a stack is called a "pop" for "popping from the stack." After an item has been "popped," its stack location is considered free and available for other use. The stack pointer points to the last-used location implying that the next (lower) location is free. Thus a stack may represent a pool of shareable temporary storage locations.

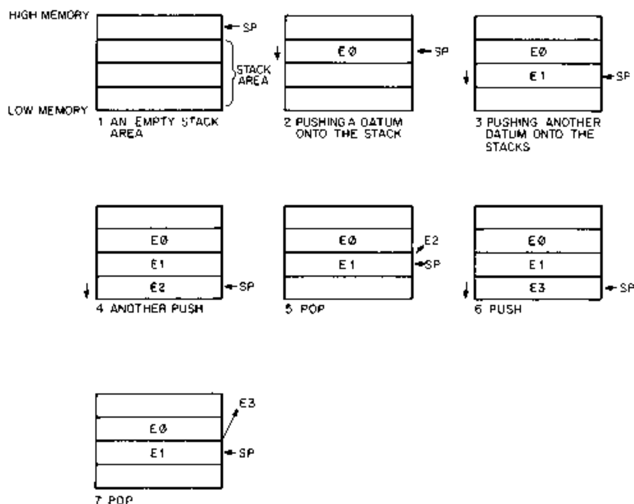


Figure 9-3: Illustration of Push and Pop Operations

As an example of stack usage consider this situation: a subroutine (SUBR) wants to use registers 1 and 2, but these registers must be returned to the calling program with their contents unchanged. The subroutine could be written as follows:

Address	Octal Code		Assembler Syntax
076322	010167	SUBR:	MOV R1,TEMP1 ; save R1
076324	000074		*
076326	010267		MOV R2,TEMP2 ;save R2
076330	000072		*
.	.		.
.	.		.
.	.		.
.	.		.
076410	016701		MOV TEMP1,R1 ;Restore R1
076412	000006		*
076414	016702		MOV TEMP2,R2 ; Restore R2
076416	000004		*
076420	000207		RTS PC
076422	000000		TEMP1: 0
076424	000000		TEMP2: 0

*Index Constants

Figure 9-4: Register Saving Without the Stack

OR: Using the Stack

Address	Octal Code		Assembler Syntax
010020	010143	SUBR:	MOV R1, —(R3) ;push R1
010022	010243		MOV R2, —(R3) ;push R2
.	.		.
.	.		.
.	.		.
.	.		.
010130	012301		MOV (R3)+,R2 ;pop R2
010132	012302		MOV (R3)+,R1 ;pop R1
010134	000207		RTS PC

Note: In this case R3 was used as a Stack Pointer

Figure 9-5: Register Saving using the Stack

The second routine uses four less words of instruction code and two words of temporary 'stack" storage. Another routine could use the same stack space at some later point. Thus, the ability to share temporary storage in the form of a stack is a very economical way to save on memory usage.

As a further example of stack usage, consider the task of managing an input buffer from a terminal. As characters come in, the terminal user may wish to delete characters from his line; this is accomplished very easily by maintaining a byte stack containing the input characters. Whenever a backspace is received a character is "popped" off the stack and eliminated from consideration. In this example, a programmer has the choice of "popping" characters to be eliminated by using either the MOVB (MOVE BYTE) or INC (INCREMENT) instructions.

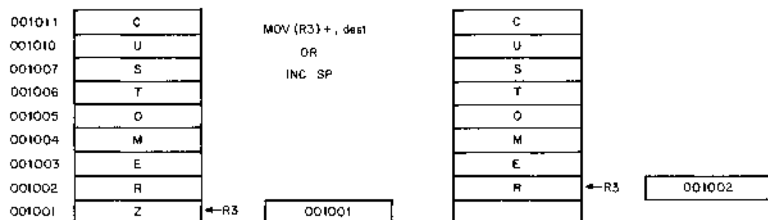


Figure 9-6: Byte Stack used as a Character Buffer

NOTE that in this case using the increment instruction (INC) is preferable to MOVB since it would accomplish the task of eliminating the unwanted character from the stack by readjusting the stack pointer without the need for a destination location. Also, the stack pointer (SP) used in this example cannot be the system stack pointer (R6) because R6 may only point to word (even) locations.

9.2 SUBROUTINE LINKAGE

9.2.1 Subroutine Calls

Subroutines provide a facility for maintaining a single copy of a given routine which can be used in a repetitive manner by other programs located anywhere else in memory. In order to provide this facility, generalized linkage methods must be established for the purpose of control transfer and information exchange between subroutines and calling programs. The PDP-11 instruction set contains several useful instructions for this purpose.

PDP-11 subroutines are called by using the JSR instruction which has the following format.

```
a general register (R) for linkage ------┐
                                   JSR R,SUBR
an entry location (SUBR) for the subroutine ┘
```

When a JSR is executed, the contents of the linkage register are saved on the system R6 stack as if a MOV reg,—(SP) had been performed. Then the same register is loaded with the memory address following the JSR instruction (the contents of the current PC) and a jump is made to the entry location specified.

Address	Assembler Syntax	Octal Code
001000	JSRR5, SUBR	004567
001002	index constant for SUBR	000060
001004		
001064	SUBR: MOV A,B	01nnmm

Figure 9-7: JSR using R5

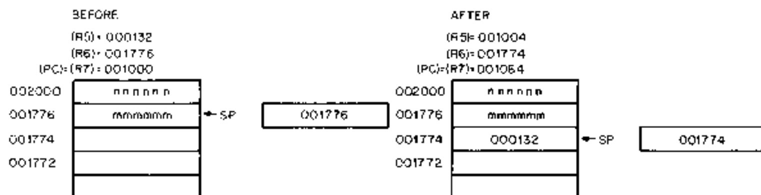


Figure 9-8: JSR

Note that the instruction JSR R6,SUBR is not normally considered to be a meaningful combination.

9.2.2 Argument Transmission

The memory location pointed to by the linkage register of the JSR instruction may contain arguments or addresses of arguments. These arguments may be accessed from the subroutine in several ways. Using Register 5 as the linkage register, the first argument could be obtained by using the addressing modes indicated by (R5),(R5)+,X(R5) for actual data, or @(R5)+, etc. for the address of data. If the autoincrement

mode is used, the linkage register is automatically updated to point to the next argument.

Figures 9-9 and 9-10 illustrate two possible methods of argument transmission.

```
Address Instructions and Data

  010400             JSR R5, SUBR
  010402             Index constant for SUBR    SUBROUTINE CALL
  010404             arg #1                     ARGUMENTS
  010406             arg #2
  .                  .
  .                  .
  .                  .
  .                  .
  .                  .
  020306   SUBR:     MOV (R5)+,R1    ;get arg #1
  020310             MOV (R5)+,R2    ;get arg #2 Retrieve Arguments
                                     from SUB
```

Figure 9-9: Argument Transmission-Register Autoincrement Mode

```
Address  Instructions and Data

010400   JSR R5, SUBR
010402   Index constant for SUBR   SUBROUTINE CALL
010404   077722                    Address of arg #1
010406   077724                    Address of arg #2
010410   077726                    Address of arg #3
.        .                         .
.        .                         .
.        .                         .
077722   arg #1
077724   arg #2                    arguments
077726   arg #3
.        .                         .
.        .                         .
.        .                         .
020306   SUBR:   MOV @(R5)+,R1 ;get arg #1
020301           MOV @(R5)+,R2 ;get arg #2
```

Figure 9-10: Argument Transmission-Register Autoincrement Deferred Mode

Another method of transmitting arguments is to transmit only the address of the first item by placing this address in a general purpose register. It is not necessary to have the actual argument list in the same general area as the subroutine call. Thus a subroutine can be called to work on data located anywhere in memory. In fact, in many cases, the operations performed by the subroutine can be applied directly to the data located on or pointed to by a stack without the need to ever actually move this data into the subroutine area.

```
Calling Program: MOV    POINTER, R1
                 JSR    PC,SUBR

SUBROUTINE       ADD    (R1) + ,(R1)   ;Add item #1 to item #2, place
                                        result in item #2, R1 points
                                        to item #2 now
                         etc.
                         or

                 ADD    (R1),2(R1)     ;Same effect as above except
                                        that R1 still points to item #1
                                        etc.
```

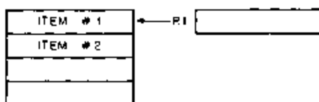


Figure 9-11: Transmitting Stacks as Arguments

Because the PDP-11 hardware already uses general purpose register R6 to point to a stack for saving and restoring PC and PS (processor status word) information, it is quite convenient to use this same stack to save and restore intermediate results and to transmit arguments to and from subroutines. Using R6 in this manner permits extreme flexibility in nesting subroutines and interrupt service routines.

Since arguments may be obtained from the stack by using some form of register indexed addressing, it is sometimes useful to save a temporary copy of R6 in some other register which has already been saved at the beginning of a subroutine. In the previous example R5 may be used to index the arguments while R6 is free to be incremented and decremented in the course of being used as a stack pointer. If R6 had been used directly as the base for indexing and not "copied," it might be difficult to keep track of the position in the argument list since the base of the stack would change with every autoincrement/decrement which occurs.

Figure 9-12: Shifting Indexed Base

However, if the contents of R6 (SP) are saved in R5 before any arguments are pushed onto the stack, the position relative to R5 would remain constant.

Figure 9-13: Constant Index Base Using "R6 Copy"

9.2.3 Subroutine Return

In order to provide for a return from a subroutine to the calling program an RTS instruction is executed by the subroutine. This instruction should specify the same register as the JSR used in the subroutine call. When executed, it causes the register specified to be moved to the PC and the top of the stack to be then placed in the register specified. Note that if an RTS PC is executed, it has the effect of returning to the address specified on the top of the stack.

Note that the JSR and the JMP Instructions differ in that a linkage register is always used with a JSR; there is no linkage register with a JMP and no way to return to the calling program.

When a subroutine finishes, it is necessary to "clean-up" the stack by eliminating or skipping over the subroutine arguments. One way this can be done is by insisting that the subroutine keep the number of arguments as its first stack item. Returns from subroutines would then involve calculating the amount by which to reset the stack pointer, resetting the stack pointer, then restoring the original contents of the register which was used as the copy of the stack pointer. The PDP-11, however, has a much faster and simpler method of performing these tasks. The MARK instruction which is stored on a stack in place of "number of argument" information may be used to automatically perform these "clean-up" chores. (For more information on the MARK instruction refer to Chapter 4.)

9.2.4 PDP-11 Subroutine Advantages

There are several advantages to the PDP-11 subroutine calling procedure.

a. arguments can be quickly passed between the calling program and the subroutine.

b. if the user has no arguments or the arguments are in a general register or on the stack the JSR PC,DST mode can be used so that none of the general purpose registers are taken up for linkage.

c. many JSR's can be executed without the need to provide any saving procedure for the linkage information since all linkage information is automatically pushed onto the stack in sequential order. Returns can simply be made by automatically popping this information from the stack in the opposite order of the JSR's.

Such linkage address bookkeeping is called automatic "nesting" of subroutine calls. This feature enables the programmer to construct fast,

efficient linkages in a simple, flexible manner. It even permits a routine to call itself in those cases where this is meaningful. Other ramifications will appear after we examine the PDP-11 interrupt procedures.

9.3 INTERRUPTS

9.3.1 General Principles

Interrupts are in many respects very similar to subroutine calls. However, they are forced, rather than controlled, transfers of program execution occurring because of some external and program-independent event (such as a stroke on the teleprinter keyboard). Like subroutines, interrupts have linkage information such that a return to the interrupted program can be made. More information is actually necessary for an interrupt transfer than a subroutine transfer because of the random nature of interrupts. The complete machine state of the program immediately prior to the occurrence of the interrupt must be preserved in order to return to the program without any noticeable effects. (i.e. was the previous operation zero or negative, etc.) This information is stored in the Processor Status Word (PS). Upon interrupt, the contents of the Program Counter (PC) (address of next instruction) and the PS are automatically pushed onto the R6 system stack. The effect is the same as if:

```
        MOV  PS  ,—(SP)                          ;Push  PS
        MOV  R7,—(SP)                            ;Push  PC
```

had been executed.

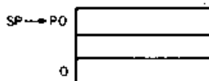
The new contents of the PC and PS are loaded from two preassigned consecutive memory locations which are called an "interrupt vector." The actual locations are chosen by the device interface designer and are located in low memory addresses of Kernel virtual space (see interrupt vector list, Appendix A). The first word contains the interrupt service routine address (the address of the new program sequence) and the second word contains the new PS which will determine the machine status including the operational mode and register set to be used by the interrupt service routine. The contents of the interrupt service vector are set under program control.

After the interrupt service routine has been completed, an RTI (return from interrupt) is performed. The two top words of the stack are automatically "popped" and placed in the PC and PS respectively, thus resuming the interrupted program.

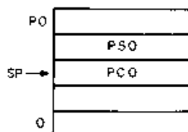
9.3.2 Nesting

Interrupts can be nested in much the same manner that subroutines are nested. In fact, it is possible to nest any arbitrary mixture of subroutines and interrupts without any confusion. By using the RTI and RTS instructions, respectively, the proper returns are automatic.

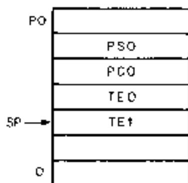
1. Process 0 is running; SP is pointing to location P0.



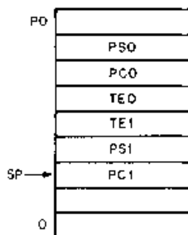
2. Interrupt stops process 0 with PC = PC0, and status = PS0; starts process 1.



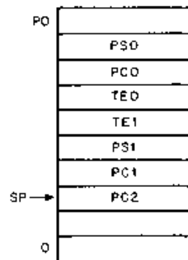
3. Process 1 uses stack for temporary storage (TE0, TE1).



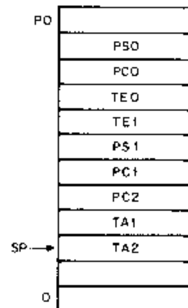
4. Process 1 interrupted with PC = PC1 and status = PS1; process 2 is started



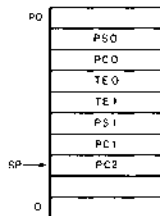
5. Process 2 is running and does a JSR R7,A to Subroutine A with PC = PC2.



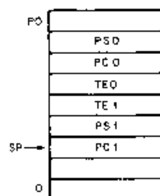
6. Subroutine A is running and uses stack for temporary storage.



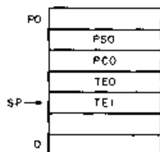
7. Subroutine A releases the temporary storage holding TA1 and TA2.



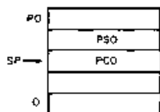
8. Subroutine A returns control to process 2 with an RTS R7,PC is reset to PC2.



9. Process 2 completes with an RTI instruction (dismisses interrupt) PC is reset to PC1 and status is reset to PS1; process 1 resumes.



10. Process 1 releases the temporary storage holding TE0 and TE1.



11. Process 1 completes its operation with an RTI is reset to PC0 and status is reset to PS0.

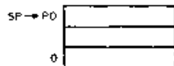


Figure 9-14: Nested Interrupt Service Routines and Subroutines

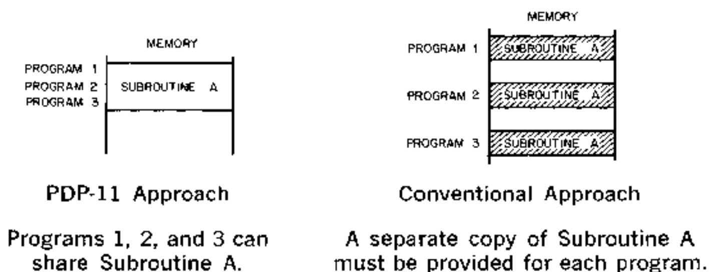
Note that the area of interrupt service programming is intimately involved with the concept of CPU and device priority levels.

9.4 REENTRANCY

Further advantages of stack organization becomes apparent in complex situations which can arise in program systems that are engaged in the concurrent handling of several tasks. Such multi-task program environ-

ments may range from relatively simple single-user applications which must manage an intermix of I/O interrupt service and background computation to large complex multi-programming systems which manage a very intricate mixture of executive and multi-user programming situations. In all these applications there is a need for flexibility and time/memory economy. The use of the stack provides this economy and flexibility by providing a method for allowing many tasks to use a single copy of the same routine and a simple, unambiguous method for keeping track of complex program linkages.

The ability to share a single copy of a given program among users or tasks is called reentrancy. Reentrant program routines differ from ordinary subroutines in that it is unnecessary for reentrant routines to finish processing a given task before they can be used by another task. Multiple tasks can be in various stages of completion in the same routine at any time. Thus the following situation may occur:



PDP-11 Approach	Conventional Approach
Programs 1, 2, and 3 can share Subroutine A.	A separate copy of Subroutine A must be provided for each program.

Figure 9-15: Reentrant Routines

The chief programming distinction between a non-shareable routine and a reentrant routine is that the reentrant routine is composed solely of "pure code," i.e., it contains only instructions and constants. Thus, a section of program code is reentrant (shareable) if and only if it is "non self-modifying," that is it contains no information within it that is subject to modification.

Using reentrant routines, control of a given routine may be shared as illustrated in Figure 9-16.

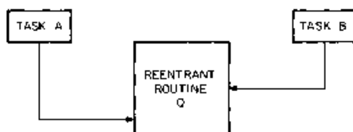


Figure 9-16: Reentrant Routine Sharing

1. Task A has requested processing by Reentrant Routine Q.
2. Task A temporarily relinquishes control (is interrupted) of Reentrant Routine Q before it finishes processing.
3. Task B starts processing in the same copy of Reentrant Routine Q.
4. Task B relinquishes control of Reentrant Routine Q at some point in its processing.
5. Task A regains control of Reentrant Routine Q and resumes processing from where it stopped.

The use of reentrant programming allows many tasks to share frequently used routines such as device interrupt service routines, ASCII-Binary conversion routines, etc. In fact, in a multi-user system it is possible, for instance, to construct a reentrant FORTRAN compiler which can be used as a single copy by many user programs.

As an application of reentrant (shareable) code, consider a data processing program which is interrupted while executing a ASCII-to-Binary subroutine which has been written as a reentrant routine. The same conversion routine is used by the device service routine. When the device servicing is finished, a return from interrupt (RTI) is executed and execution for the processing program is then resumed where it left off inside the same ASCII-to-Binary subroutine.

Shareable routines generally result in great memory saving. It is the hardware implemented stack facility of the PDP-11 that makes shareable or reentrant routines reasonable.

A subroutine may be reentered by a new task before its completion by the previous task as long as the new execution does not destroy any linkage information or intermediate results which belong to the previous programs. This usually amounts to saving the contents of any general purpose registers, to be used and restoring them upon exit. The choice of whether to save and restore this information in the calling program or the subroutine is quite arbitrary and depends on the particular application. For example in controlled transfer situations (i.e. JSR's) a main program which calls a code-conversion utility might save the contents of registers which it needs and restore them after it has regained control, or the code conversion routine might save the contents of registers which it uses and restore them upon its completion. In the case of interrupt service routines this save/restore process must be carried out by the service routine itself since the interrupted program has no warning of an impending interrupt. The advantage of using the stack to save and restore (i.e. "push" and "pop") this information is that it permits a program to isolate its instructions and data and thus maintain its reentrancy.

In the case of a reentrant program which is used in a multi-programming environment it is usually necessary to maintain a separate R6 stack for each user although each such stack would be shared by all the tasks of a given user. For example, if a reentrant FORTRAN compiler is to be shared between many users, each time the user is changed,

R6 would be set to point to a new user's stack area as illustrated in Figure 9-17.

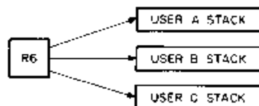


Figure 9-17: Multiple R6 Stack

9.5 POSITION INDEPENDENT CODE—PIC

Most programs are written with some direct references to specific addresses, if only as an offset from an absolute address origin. When it is desired to relocate these programs in memory, it is necessary to change the address references and/or the origin assignments. Such programs are constrained to a specific set of locations. However, the PDP-11 architecture permits programs to be constructed such that they are not constrained to specific locations. These Position Independent programs do not directly reference any absolute locations in memory. Instead all references are "PC-relative" i.e. locations are referenced in terms of offsets from the current location (offsets from the current value of the Program Counter (PC)). When such a program has been translated to machine code it will form a program module which can be loaded anywhere in memory as required.

Position Independent Code is exceedingly valuable for those utility routines which may be disk-resident and are subject to loading in a dynamically changing program environment. The supervisory program may load them anywhere it determines without the need for any relocation parameters since all items remain in the same positions relative to each other (and thus also to the PC).

Linkages to program routines which have been written in position independent code (PIC) must still be absolute in some manner. Since these routines can be located anywhere in memory there must be some fixed or readily locatable linkage addresses to facilitate access to these routines. This linkage address may be a simple pointer located at a fixed address or it may be a complex vector composed of numerous linkage information items.

9.6 CO-ROUTINES

In some situations it happens that two program routines are highly interactive. Using a special case of the JSR instruction i.e., JSR PC, @(R6)+ which exchanges the top element of the Register 6 processor stack and the contents of the Program Counter (PC), two routines may be permitted to swap program control and resume operation where they stopped, when recalled. Such routines are called "co-routines." This control swapping is illustrated in Figure 9-18.

Routine #1 is operating, it then executes:

```
MOV #PC2,-(R6)

JSR PC,@(R6)+
```

with the following results:

1) PC2 is popped from the stack and the SP autoincremented
2) SP is autodecremented and the old PC (i.e. PC1) is pushed
3) control is transferred to the location PC2 (i.e. routine #2)

Routine #2 is operating, it then executes:

```
JSR PC, @(R6)+
```

with the result the PC2 is exchanged for PC1 on the stack and control is transferred back to routine #1.

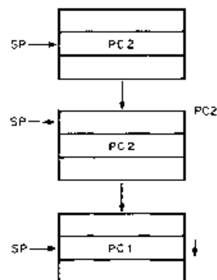


Figure 9-18—Co-Routine Interaction

CHAPTER 10

HIGH SPEED I/O CONTROLLERS

10.1 SYSTEM PERFORMANCE

To support the speed, power, and data reliability features of the PDP-11/70 central processor and memory system, DIGITAL offers a wide range of high-performance, mass-storage peripheral options. These secondary storage disk and magnetic tape systems interface to the central processor through optimized high-speed controllers and dedicated data paths to provide high system throughput. Since the control and interfacing of these high-performance peripherals is an integral part of the PDP-11/70 architecture, increased input/output capabilities are achieved. These peripherals become a vital part of the PDP-11/70 system.

10.2 HIGH-SPEED, MASS STORAGE PERIPHERALS

There are, currently, 3 high-performance peripherals that can take advantage of interfacing to the PDP-11/70 through its high-speed controllers and high data rate bus.

a) RS04 (and RS03) Fixed Head Disk
b) RP04 Disk Pack
c) TU16 Magnetic Tape Unit

10.2.1 Fixed Head Disk

The RS03 and RS04 fixed-head disks have been designed for applications requiring fast, reliable, on-line storage. With an average access time of 8.5 milliseconds and a transfer rate of 2 microseconds per word (4 μsec for RS03), the disks increase throughput substantially for timesharing applications which involve significant amounts of program swapping. Phase lock loop reading techniques and CRC error detection make these disk systems ideal for real-time data acquisition and control systems requiring a high level of reliability.

The RS03 fixed-head disk drive has a storage capacity of 256K words, and the RS04 has a storage capacity of 512K words. The disks are expandable by adding either RS03 or RS04 drives, up to a total of eight drives per controller.

SPECIFICATIONS

	RS03	**RS04 (when different)**
Storage medium	Fixed-head disk	
Capacity/disk	262,144 words (256K)	512K words
Data transfer speed	4 μsec/word	2 μsec/word
Average access time (1/2 rev)	8.5 msec	
Minimum access time	6.4 μsec	

Disk rotation speed	3600 RPM (3000 RPM at 50 Hz)
Disks/control, maximum	8

10.2.2 Disk Pack
The RP04 is a mass storage system offering low cost per bit and high performance. Each disk pack has a capacity of 44 million 16-bit words expandable to 8 disk pack drives in a system. The removable disk pack offers the flexibility of unlimited off-line storage capacity.

On multi-drive systems, positioning operations can be overlapped for efficiency. While one drive is reading or writing, one or more drives can be positioning to a new cylinder for the next transfer.

The RP04 operates at a transfer rate of 400,000 words per second (2.5 microseconds per word).

The disk drive is a high-performance device, featuring direct access and single head per surface. It enables the data processing system to store or retrieve information at any location on a rotating disk pack.

SPECIFICATIONS

Storage medium:	Disk pack
Capacity/pack:	43,980,288 words
Data transfer speed:	2.5 μsec/word
Time for 1/2 revolution:	8.3 msec
Disk rotation speed:	3600 RPM
Drives/control, maximum:	8
One cylinder seek:	7 msec
Average seek:	28 msec
Maximum seek:	50 msec

10.2.3 Magnetic Tape
The TU16 is a fully integrated, high-performance magnetic tape storage system that uses standard recording formats, with densities of 1600 and 800 bits per inch, selectable under program control. Reading and writing are performed at 45 inches/second. Since the industry standard format is used, data may be easily transferred between computers.

Reading can be performed while tape is moving in the forward or reverse direction, but writing occurs only in forward. The control unit can move the tape to new positions in forward or reverse.

Tape motion is controlled by vacuum columns and a servo-controlled single capstan. Long tape life is possible because the only contact with the oxide surface is at the magnetic head and at a rolling contact on one low-friction, low-inertia bearing.

Main Specifications

Storage medium:	½-inch wide magnetic tape (industry std)
Capacity/tape reel:	32 million characters (at 1600 bpi)
Data transfer speed:	72,000 characters/sec., max.
Drives/control:	8, max.

Data Organization

Number of tracks:	9
Recording density:	800 or 1600 bits/inch, program selectable
Interrecord gap:	0.50 inches, min
Recording method:	NRZI for 800 bpi, phase encoded for 1600 bpi

Tape Motion

Read/write speed:	45 inches/sec.
Rewind speed:	150 inches/sec.
Rewind time:	3 minutes, typical

Tape Characteristics

Length:	2,400 feet, max.
Type:	Mylar base, iron oxide coated
Reel diameter:	10½ inches, max.
Handling:	direct-drive reel motors, servo-controlled single capstan, vacuum tape buffer changers with constant tape tension

10.3 HIGH-SPEED CONTROLLERS

Mounting Space

The PDP-11/70 CPU assembly provides dedicated, pre-wired space for up to 4 high-speed I/O controllers. Refer to Figure 10-1. DC power for the controllers is derived from the cabinet power supply.

Interfacing

Each group of mass storage peripherals communicates with its high-speed controller through a separate high-speed I/O bus. This I/O bus consists of a set of 56 signals for data, control, status, and parity. High transfer rate is achieved by using synchronous block transfer of data simultaneously with asynchronous control information. The controller contains an 8-word data buffer.

Data is transferred in a Direct Memory Access (DMA) mode. An internal 32-bit wide data bus transfers 4 bytes in parallel between memory and the high-speed controllers. The Priority Arbitration logic within the cache memory controls the timing of data transfers; but the cache itself is not used for data storage. Data transfers are between main (core) memory and the mass storage peripheral. The cache is not affected, except that on a write hit from the I/O Bus to memory, the valid bit is cleared for that particular 2-word block within the cache. In this way, the affected areas of the cache are flagged as having incorrect data, but main memory always contains the correct, updated information.

The UNIBUS plays a subordinate role with respect to the high-speed controllers. The UNIBUS is used:

a) to supply control and status information
b) to generate an interrupt request (by the controller)

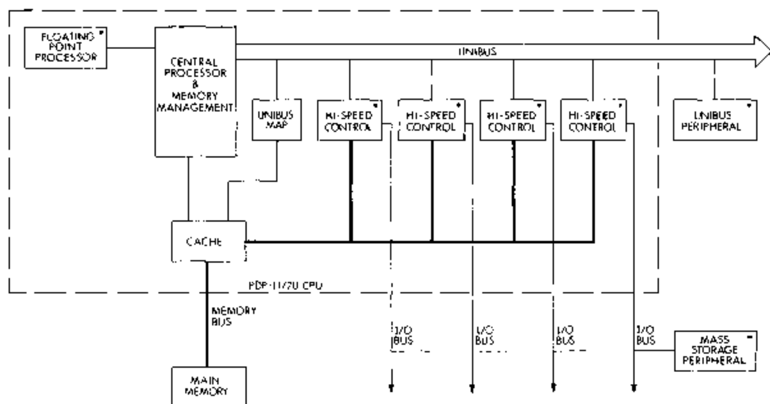


Figure 10-1 PDP-11/70 Block Diagram

The UNIBUS is not used for data transfer.

The registers within the controller (which can be read and written directly) are addressed from the UNIBUS. In a typical DMA transfer, the registers would first be loaded with the following data:

a) number of words to be transferred
b) starting address in memory for data transfers
c) control information specifying the device and type of operation.

Then the GO command would be issued (to the register), and data transfer would proceed without CPU intervention.

Increased Data Transfer Rate
The architecture of the PDP-11/70 allows overlapping of some operations, thereby providing faster program execution speed. CPU and UNIBUS read hits with the cache memory are overlapped with mass storage device reads from main memory. It is possible to overlap the read cycles of several mass storage devices.

Parity
Parity is generated and checked in the system for both data, and address and control information, to ensure the integrity of the information transferred. The RHCS3 register in the controller is used to indicate the occurrence of parity errors during memory transfers.

10.4 REGISTERS
The controller contains 6 local registers, plus part of 1 more which is shared with the mass-storage device. Other registers needed by the

particular mass storage system and device are contained in the device itself. Appendix B contains information about the mass storage device registers.

Controller Registers

RHCS1 Control and Status 1 (partial)

RHWC Word Count

RHBA Bus Address (Main Memory Bus)

RHBAE Bus Address Extension (Main Memory Bus)

RHCS2 Control and Status 2

RHCS3 Control and Status 3

RHDB Data Buffer (Maintenance)

10.5 CONTROLLER REGISTERS

Control and Status 1 Register (RHCS1)

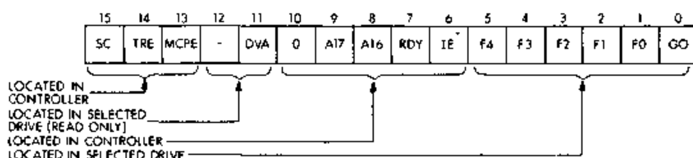
This register is utilized by both the controller and the mass storage device to store the device commands and hold operational status. Register bits 0 thru 5, 11, and 12 are dedicated for use by the drive and are physically located in each drive attached to the controller. When reading or writing this register, the selected drive (indicated by bits 2 thru 0 in the RHCS2 register) will respond in those bit positions.

When the program reads, writes a word, or writes the low byte of this register, a register cycle will be initiated to the selected drive over the high-speed I/O bus. If the unit selected does not exist or respond, an NED (non-existing drive) error will result. The program may, however, write the upper byte of this register without regard to the unit selected and without affecting any drive.

Register bits 0 thru 5 indicate the command to be performed and are actually stored in the selected drive. The controller will always interrogate the command code being passed to the drive by the program and will prepare for the appropriate memory cycle required by data transfer operations. Data transfer command codes are designated by 51_8 thru 77_8 (always odd since the GO bit must be asserted to execute the function) and will cause the controller to become busy (RDY negated) until the completion of the operation. When the controller is busy, no further data transfer commands may be issued (see PGE bit 10 in RHCS2). Non-data transfer commands, however, may be issued at any time and to any drive which is not busy.

While a data transfer is in progress, unit select bits U(02:00) in RHCS2 may be changed by the program in order to issue a non data transfer command to another drive. This will not affect the data transfer.

When a non-data transfer command code is written into RHCS1 while a data transfer is taking place, only the even (low) byte of RHCS1 should be written. This will prevent the program from unintentionally changing the A16 and A17 status bits if the transfer is completed just before the register is written. (While the RDY bit is negated, the controller prevents program modification of these control bits even when the write is done to the odd byte.)



Control and Status 1 Bit Usage

BIT	SET BY/CLEARED BY	REMARKS
15 SC Special Condition Read Only	Set by TRE or Attention or MCPE. Cleared by Unibus INIT, controller clear, or by removing the Attention condition.	SC = TRE + ATTN + MCPE. Attention occurs when any drive has a) an error condition, b) a change in status or c) completed a function requiring action by the program (other than data transfer.
14 TRE TRansfer Error Read/Write	Set by DLT or WCE or PE or NED or NEM or PGE or MXF or MDPE or a drive error during a data transfer. Cleared by Unibus INIT, controller clear, error clear (the action of writing a 1 in the TRE bit), or by loading a data transfer command with GO set.	TRE = DLT + WCE + PE + NED + NEM + PGE + MXF + MDPE + (EXCP•EBL)
13 MCPE Mass I/O Bus Control Parity Error Read Only	Set by a parity error on the control section of the I/O bus when reading a remote register (located in the drive). Cleared by Unibus INIT, controller clear, error clear, or by loading a data transfer command with GO set.	Parity errors which occur on the control bus when writing a drive register are detected by the drive. Parity checking occurs at the completion of the register cycle (an MCPE when reading the RHCS1 register would not be indicated on the same cycle.
12 Reserved for use by the Drive Read Only	Always read as 0 if not implemented by the selected drive.	

BIT	SET BY/CLEARED BY	REMARKS
11 DVA DriVe Available Read Only	Implemented by the drive. Set when the selected drive is available to the controller.	Used in dual-port drive applications. Always a 1 in single port drives.
10 Not used	Always read as 0.	
9 A17 8 A16 Bus Address Extension Bits Read/Write	Upper address extension bits of the BA register. Cleared by Unibus INIT, controller clear, or by writing 0's in these bit positions.	These bits cannot be modified by writing to the RHCS1 register while the controller is busy (RDY negated). Incremented by a carry from the RHBA register during data transfers to/from memory. These bits can also be set/cleared thru the RHBAE register.
7 RDY Ready Read only	Indicates controller status. When set the controller will accept any command. When cleared the controller is performing a data transfer command and will allow only non-data transfer commands to be executed.	The assertion of RDY (transfer complete or TRE) will cause an interrupt if IE = 1.
6 IE Interrupt Enable Read/Write	Control bit which can be set under program control. When IE = 1, an interrupt may occur due to RDY or Attention or MCPE being asserted. Cleared by Unibus INIT, controller clear, or automatically cleared when an interrupt is recognized by the CPU.	A program-controlled interrupt may occur by writing 1's into IE and RDY at the same time. This bit can be set/cleared thru the RHCS3 register.
5-0 F4-F0 and GO Read/Write	F4-F0 are function (command) code control bits which determine the action to be performed by the controller and/or drive. The GO bit must be set in order to execute the command. The GO bit	The function code bits are stored in the selected drive. Only data transfer commands (defined as F4•(F3 + F2)•GO will cause the controller to become busy (RDY negated). All other command codes

BIT	SET BY/CLEARED BY	REMARKS
	is reset by the drive at the end of the operation.	are ignored by the controller.

Function Code Table

F4	F3	F2	F1	F0	
0	0	0	0	0	Reserved for drive related commands. No controller action taken.
		thru			
1	0	0	1	1	
1	0	1	0	0	Write Check commands. Memory data compared with drive data in controller. Memory address increments.
1	0	1	0	1	
1	0	1	1	0	
1	0	1	1	1	Write Check command. Memory address decrements
1	1	0	0	0	Write commands. Memory data written into drive. Memory address increments.
1	1	0	0	1	
1	1	0	1	0	
1	1	0	1	1	Write command. Memory address decrements.
1	1	1	0	0	Read commands. Drive data written into Memory. Memory address increments.
1	1	1	0	1	
1	1	1	1	0	
1	1	1	1	1	Read command. Memory address decrements.

Word Count Register (RHWC)

This register is loaded by the program with the two's complement of the number of words to be transferred. During a data transfer, it is incremented by 1 each time a word is transmitted to or from memory.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WC15	WC14	WC13	WC12	WC11	WC10	WC09	WC08	WC07	WC06	WC05	WC04	WC03	WC02	WC01	WC00

Word Count Register Bit Usage

BIT	SET BY/CLEARED BY	REMARKS
WC(15:00) Word Count Read/Write	Set by the program to specify the number of words to be transferred (Two's complement form.) This register is cleared only by writing 0's into it.	Incremented for each word transferred to/from memory.

Bus Address Register (RHBA)
This register is loaded by the program to specify the lower 16 bits of the starting memory address to which data transfers will take place. The RHBA and RHBAE registers combine to form the complete 22 bit memory address.

During a data transfer this register is incremented (decremented for specific function codes) by 2 each time a word is transmitted to or from memory. If the BAI (Bus Address Increment Inhibit) bit (bit 03 of RHCS2) is set, the incrementing (or decrementing) of the RHBA register is inhibited and all transfers take place to or from the starting memory address.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A15	A14	A13	A12	A11	A10	A09	A08	A07	A06	A05	A04	A03	A02	A01	0

Bus Address Register Bit Usage

BIT		SET BY/CLEARED BY	REMARKS
15:01	A (15:01) Bus Address Read/Write	Loaded by the program to specify the starting memory address of a data transfer operation. Cleared by Unibus INIT or controller clear	The RHBA register is incremented (or decremented) by 2 whenever a word is transmitted to or from memory.
00	Not Used	Always read as a 0	

Bus Address Extension Register (RHBAE)
The RHBAE register contains the upper 6 bits of the memory address and combine with the lower 16 bits located in RHBA to form the complete 22 bit address. This register should be loaded by the program in conjunction with the RHBA register to specify the starting memory address of a data transfer operation. The six bit field is incremented (decremented for specific function codes) each time a carry (borrow) occurs from the RHBA register during memory transfers.

Address bits A16 and A17 can also be set or cleared thru the RHCS1 register. If an address extension field is written into RHBAE, the program should ensure that A16 and A17 are not altered when a command is loaded into RHCS1. This can be accomplished by either loading the command with a write low byte instruction to RHCS1 or by ensuring the proper value appears in the A16 and A17 bit positions of RHCS1.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	A21	A20	A19	A18	A17	A16

Bus Address Extension Register Bit Usage

BIT		SET BY/CLEARED BY	REMARKS
15:06	Not Used	Always read as a 0	
05:00	A(21:16) Bus Address Read/Write	Loaded by the program to specify the starting memory address of a data transfer operation. Cleared by Unibus INIT or controller clear.	The RHBAE register is incremented (or decremented) each time a carry out (borrow out) of RHBA occurs. A16 and A17 can also be set or cleared thru the RHCS1 register.

Control and Status 2 Register (RHCS2)
This register indicates the status of the controller and contains the drive unit number U(2:0). The unit number specified in bits 2 thru 0 of this register indicates which drive is responding when registers are addressed which are located in a drive.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DLT	WCE	PE	NED	NEM	PGE	MXF	MDPE	OR	IR	CLR	PAT	BAT	U2	U1	U0

Control and Status 2 Register Bit Usage

BIT		SET BY/CLEARED BY	REMARKS
15	DLT Data LaTe Read only	Set when the controller is unable to supply a data word during a write operation or accept a data word during a Read or Write-check operation at the time the drive demands a transfer. Cleared by Unibus INIT, controller clear, error clear, or loading a data transfer command with GO set.	DLT causes TRE. Buffering is 8 words deep in the controller and a DLT error indicates a severely overloaded system.
14	WCE Write Check Error Read only	Set when the controller is performing a write-check operation and a word on the drive does	WCE causes TRE. If a mismatch is detected during a Write-check command execution

BIT	SET BY/CLEARED BY	REMARKS
	not match the corresponding word in memory. Cleared by Unibus INIT, controller clear, error clear, or loading a data transfer command with GO set.	the transfer terminates and the WCE bit is set. The mismatched data word from the drive is displayed in the data buffer (RHDB).
13 PE Parity Error Read only	Set if a parity error occurred between memory and the controller during a memory transfer. Cleared by Unibus INIT controller clear, error clear, or loading a data transfer command with GO set.	PE = APE + DPEOW + DPEEW
12 NED Non-Existent Drive Read only	Set when the program reads or writes a register in a drive (selected by U(02:00) which does not exist or is powered down. (The drive fails to assert TRA within 1.5 μs after assertion of DEM. Cleared by Unibus INIT, controller clear, error clear, or loading a data transfer command with GO set.	NED causes TRE.
11 NEM Non-Existent Memory Read only	Set when the controller is performing a DMA transfer and the memory address specified in RHBA is non-existent. Cleared by Unibus INIT, controller clear, error clear, or loading a data transfer command with GO set.	NEM causes TRE to set.
10 PGE Program Error Read only	Set when the program attempts to initiate a data transfer operation while the controller is currently performing one. Cleared by Unibus INIT, controller clear, error clear, or loading a data transfer command with GO set.	PGE causes TRE to set. The data transfer command code is inhibited from being written into the drive.

BIT	SET BY/CLEARED BY	REMARKS
09 MXF Missed Transfer Read only	Set if the drive does not respond to a data transfer command within 500 μsec. Cleared by Unibus INIT, controller clear, error clear, or loading a data transfer command with GO set.	MXF causes TRE to set. This error occurs if a data transfer command is loaded into a drive which has ERR set, or if the drive fails to initiate the command for any reason (such as parity error or illegal function.)
08 MDPE Mass I/O Bus Data Parity Error Read only	Set when a parity error occurs on the data section of the I/O bus while doing a read or write-check operation. Cleared by Unibus INIT, controller clear, error clear, or loading a data transfer command with GO set.	MDPE causes TRE. Parity errors on the data bus during write operations are detected by the drive.
07 OR Output Ready Read only	Set when a word is present in RHDB and can be read by the program, cleared by Unibus INIT, controller clear, or by reading DB.	Serves as a status indicator for diagnostic check of the data buffer.
06 IR Input Ready Read only	Set when a word may be written in the RHDB register by the program. Cleared when the data buffer is full (contains 8 words).	Serves as a status indicator for diagnostic check of the data buffer.
05 CLR Controller Clear Write only	When a 1 is written into this bit, the controller and all drives are initialized.	Unibus INIT also causes Controller Clear to occur.
04 PAT Parity Test Read/Write	While PAT is set, the controller generates even parity on both the Control and Data sections of the I/O bus. When clear, odd parity is generated. Cleared by Unibus INIT or controller clear.	While PAT is set, the controller checks for even parity received on the Data Bus but not on the Control Bus.

BIT	SET BY/CLEARED BY	REMARKS
03 BAI Unibus Address Increment Inhibit Read/Write	When BAI is set, the controller will not increment the BA register during a data transfer. This bit cannot be modified while the controller is doing a data transfer (RDY negated). Cleared by Unibus INIT or controller clear.	When set during a data transfer, all data words are read from or written into the same memory location.
02-00 U(2:0) Unit Select (2:0) Read/Write	These bits are written by the program to select a drive. Cleared by Unibus INIT or controller clear.	The unit select bits can be changed by the program during data transfer operations without interfering with the transfer.

Control and Status 3 Register (RHCS3)
The RHCS3 register contains parity error information associated with the memory bus. Bit position 13 of the RHCS2, PE, indicates that a parity error occurred during the memory transfer. Bits 15 thru 13 of RHCS3 further localize the error for diagnostic maintenance. In addition, bits 3 thru 0 provide the diagnostic program the ability to invert the sense of parity check and thereby verify correct operation of the parity circuits.

An Interrupt Enable bit in the RHCS3 register allows the program to enable interrupts without writing into a drive register as previously described. This bit also appears in the RHCS1 register for program compatibility and can be set or cleared by writing into either register.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
APE	DPE OW	DPE EW	WCE OW	WCE EW	DBL	0	0	0	IE	0	0	IPCK 3	IPCK 2	IPCK 1	IPCK 0

Control and Status 3 Bit Usage

BIT	SET BY/CLEARED BY	REMARKS
15 APE Address Parity Error Read Only	Set if the address parity error line indicates that the memory detected a parity error on address and control information during a memory transfer. Cleared by Unibus Init, controller clear, error clear, or loading a	APE causes PE, bit 13 of RHCS2. When an APE error occurs the RHBA and RHBAE registers contain the address +4 of the double word address at which the error occurred during a dou-

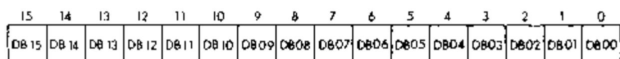
BIT	SET BY/CLEARED BY	REMARKS
	data transfer command with GO set.	ble word operation or the address +2 during a single word operation.
14, 13 DPE, OW, EW Data Parity Error Odd Word, Even Word Read Only	Set if a parity error is detected on data from memory when the control is performing a Write or Write Check command. Cleared by Unibus Init, controller clear, error clear, or loading a data transfer command with GO set.	DPE causes PE, bit 13 of RHCS2. When a DPE error occurs, the RHBA and RHBAE registers contain the address +4 of the double word address at which the error occurred during a double word operation or the address +2 during a single word operation.
12, 11 WCE OW, EW Write Check Error Odd word, Even word, Read only	Set when data fails to compare between memory and the drive. Cleared by Unibus Init, controller clear, error clear, or loading a data transfer command with the GO bit set.	Causes WCE, bit 14 of RHCS2. The word read from the drive which did not compare is locked in the data buffer and can be examined by reading the RHDB register.
10 DBL DouBle word Read Only	Set if the last memory transfer was a double word operation. Cleared by Unibus Init, controller clear or loading a data transfer command with GO set.	
9-7 Not Used	Always read as a 0	
6 IE Interrupt Enable Read/Write	IE is a control bit which can be set under program control. When IE = 1, an interrupt may occur due to RDY or SC being asserted. Cleared by Unibus Init, controller clear, or automatically cleared when an interrupt is recognized by the CPU. When a 0 is written into IE by the program, any pending interrupts are cancelled.	This bit can also be set or cleared by writing into RHCS1 register. If written thru RHCS3 register write operation is not performed into a drive register simultaneously.

BIT	SET BY/CLEARED BY	REMARKS
5-4 Not Used	Always read as a 0	
3-0 IPCK (3:0) Invert Parity Check (3:0) Read/Write	These bits are written by the program to control the data parity detection logic. When set inverse parity is checked with data during memory transfers of Write and Write Check operations.	Parity control is provided for each byte in double word addresses. i.e. IPCK 0—Even Word, Even Byte IPCK 1—Even Word, Odd Byte IPCK 2—Odd Word, Even Byte IPCK 3—Odd Word, Odd Byte

Data Buffer Register (RHDB)
This register provides a maintenance tool to check the data buffer in the controller. A total of 8 words is accepted before the data buffer becomes full. Successive reads from DB read out words in the same order in which they were entered into the data buffer.

The IR (input ready) and OR (output ready) status indicators in the RHCS2 register are provided so that the programmer can determine when words can be read from or written into the RHDB. IR should be asserted before attempting a write into DB; OR should be asserted before attempting a read from DB.

The RHDB register can be read and written only as an entire word. Any attempt to write a byte will cause an entire word to be written. Reading the DB register is a "destructive read-out" operation: the top data word in the data buffer is removed by the action of reading DB, and a new data word (if present) replaces it a short time later. Conversely, the action of writing the DB register does not destroy the "contents" of DB; it merely causes one more data word to be inserted into the data buffer (if it was not full).



Data Buffer Bit Usage

BIT	DATA BUFFER BIT ASSIGNMENTS	REMARKS
15-00 DB(15:00) Data Buffer (15:00) Read/Write	When read, the contents of OBUF (internal register) are delivered. Upon completion of the read the next sequential word in the buffer	Used by the program for diagnostic purposes. When the register is written into, IR is cleared until the DB is ready to accept a

BIT	DATA BUFFER BIT ASSIGNMENTS	REMARKS
	will be clocked into OBUF.	new word. When the register is read, it will cause OR to be cleared until a new word is ready. During a Write Check Error condition the data word read from the disk which did not compare with the corresponding word in memory is frozen in RHDB for examination by the program.

NOTE

Appendix B contains register diagrams for each High Speed I/O subsystem. Detailed descriptions of bit assignments for each I/O device register may be found in the PDP-11 Peripherals Handbook, 1975 edition.

CHAPTER 11

CONSOLE OPERATION

11.1 INTRODUCTION

The PDP-11/70 console allows direct control of the computer system. It contains a power switch for the CPU, which is also usually 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 operation, system control, and maintenance. Debugging and detailed tracing of operations can be accomplished by having the computer execute single instructions or single cycles. Contents of all locations can be examined, and data can be entered manually from the console switches.

11.2 GENERAL

The PDP-11/70 Operator's Console provides the following facilities:

a) Power Switch (with a key lock)

b) ADDRESS Register display (22 bits)

c) DATA Register display (16 bits), plus Parity Bit Low Byte, & Parity Bit High Byte

d) Switch Register (22 switches)

e) Error Lights
   - ADRS ERR (Address Error)
   - PAR ERR (Parity Error)

f) Processor State Lights (7 indicators)
   - RUN
   - PAUSE
   - MASTER
   - USER
   - SUPERVISOR
   - KERNEL
   - DATA

g) Mapping Lights
   - 16 BIT
   - 18 BIT
   - 22 BIT

h) ADDRESS Display Select Switch (8 positions)
   - USER I (Virtual)
   - USER D (Virtual)
   - SUPER I (Virtual)
   - SUPER D (Virtual)
   - KERNEL I (Virtual)
   - KERNEL D (Virtual)
   - PROG PHY (Program Physical)
   - CONS PHY (Console Physical)

i) DATA Display Select Switch (4 positions)
   - DATA PATHS
   - BUS REGISTER
   - ADRS FPP/CPU
   - DISPLAY REGISTER

j) LAMP TEST SWITCH

k) Control Switches
   - LOAD ADRS
   - EXAM (Examine)
   - DEP (Deposit)
   - CONT (Continue)
   - ENABLE/HALT
   - S INST/S BUS CYCLE (Single Instruction/Single Bus Cycle)
   - START

11.3 STARTING AND STOPPING

Starting
Once power is on, execution can be started by placing the ENABLE/HALT Switch in the ENABLE position, putting the starting address in the Switch Register, and depressing the LOAD ADRS Switch. Verify in the Address Display Lights that the address was entered correctly, then depress the START Switch. The computer system will be cleared and will then start running. Once execution has begun, depressing the START Switch again has no effect.

If the system needs to be initialized but execution is not wanted, the START Switch should be depressed **while** the HALT/ENABLE Switch is in the HALT position.

Stopping
Set the ENABLE/HALT Switch to the HALT position. The computer will stop execution, but the contents of all memory locations will be retained. The switch can then be set to the ENABLE position with no effect on the system.

NOTE

NPR's are still serviced after a halt from the console if S BUS CYCLE is disabled.

Continuing
After the computer has been stopped, execution can be resumed from the point at which it was halted by using the CONT (Continue) Switch. The function of the CONT Switch depends on the position of the ENABLE/HALT Switch:

ENABLE (up)	CPU resumes normal execution.
HALT (down)	The mode is used for debugging purposes and forces execution of only a single instruction or a single bus cycle. This is discussed in Section 11.7.

11.4 REFERENCING MEMORY

Unmapped References
When performing unmapped memory references from the console, the

Address Select Switch must be set to CONS PHY. This means that the 22-bit address entered in the Switch Register should be the physical address desired. To examine a memory location, depress the LOAD ADRS Switch and then the EXAM Switch. The address referenced will appear in the Address Display Lights. The Data Select Switch should be selecting DATA PATHS, and the contents of that location are displayed in the Data Display Lights. To deposit information into a memory location, depress the LOAD ADRS Switch, then enter the desired data in the Switch Register and raise the DEP Switch. The DATA Select Switch should be in the DATA PATHS position, and the deposited information will appear in the Data Display Lights.

Mapped References

Sometimes when software is running with Memory Management enabled the physical addresses generated are not known. This makes examining and depositing memory locations more difficult. For this reason, the 6 positions KERNEL I through USER D of the Address Select Switch are provided. When doing a memory reference the low order 16 bits of the Switch Register are considered to be a Virtual Address and are relocated by Memory Management using the set of PAR/PDR's indicated by the Address Select Switch.

To examine a memory location, depress the LOAD ADRS Switch and the EXAM Switch. The data Select Switch should be selecting DATA PATHS, and the contents of that location are displayed in the DATA Display Lights. To deposit information ino a memory location, depress the LOAD ADRS Switch, then enter the desired data in the Switch Register and raise the DEP Switch. The Data Select Switch should be in the DATA PATHS position, and the deposited information will appear in the DATA DISPLAY Lights.

The PROG PHY (Program Physical) position of the Address Select Switch is used as a debugging tool. After an examine or deposit has been performed on a virtual address, changing the Address Select Switch to select PROG PHY will display the Physical Address generated by Memory Management in the Address Display Lights. Using the PROG PHY position in any other way will produce meaningless results.

NOTE

An EXAM or DEP operation which causes an addressing error (ADRS ERR or PAR ERR) will be aborted and must be corrected by performing a new LOAD ADRS operation with a valid address.

11.5 STEP OPERATIONS

Performing more than one EXAM operation in a row or more than one DEP operation in a row results in a STEP-operation. Depressing the EXAM Switch after a previous examine of a location displays the contents of the next location in memory. Raising the DEP Switch after a previous deposit into a memory location causes the current contents of the Switch Register to be deposited into the next location in memory.

In each case, the Address Display is updated by 2 to hold the value of the now current address. This allows consecutive EXAM operations and

consecutive DEP operations without the use of the LOAD ADRS Switch. An EXAM-STEP or DEP-STEP operation will not cross a 32K word memory block boundary.

NOTE

The EXAM and DEP Switches are coupled to enable an EXAM—DEP—EXAM sequence to be carried out on a location without having to do extra LOAD ADRS operations. The following example deposits values into consecutive memory locations.

Operation (Activate Switch)	Location shown in ADDRESS Display
LOAD ADRS	X
EXAM	X
DEP	X
EXAM	X
EXAM (result is EXAM—STEP)	X + 2
DEP	X + 2
EXAM	X + 2

11.6 GENERAL REGISTERS

The General Registers can be examined and deposited using the EXAM and DEP Switches provided the previous LOAD ADRS operation loaded the Address Display with a "register address."

Address	Register
17 777 700	Register 0 (Set 0)
|	|
17 777 705	Register 5 (Set 0)
17 777 706	Register 6, Kernel Mode
17 777 707	Program Counter
17 777 710	Register 0 (Set 1)
|	|
17 777 715	Register 5, (Set 1)
17 777 716	Register 6, Supervisor Mode
17 777 717	Register 6, User Mode

Examining and depositing into General Register Addresses is independent of the Address Select Switch. It is not possible to be mapped to a General Register.

EXAM-STEP and DEP-STEP operations can be performed on the General Registers, similar to that for memory locations, except that:

a) ADDRESS Display is incremented by 1 (instead of 2)

b) The STEP after address 17 777 717 is 17 777 700, such that the addresses are looped.

c) It is not possible to STEP up to the first General Register (17 777 700) from 17 777 676

11.7 SINGLE INSTRUCTION/SINGLE BUS CYCLE

Once the machine is halted, a useful debugging tool is being able to execute code, a small segment at a time. The S INST/ S BUS CYCLE (Single Instruction/Single Bus Cycle) Switch provides that capability. The ENABLE/HALT Switch must be in the HALT position. To start execution of a segment depress the CONT Switch. How much is executed is a function of the S INST/S BUS CYCLE Switch.

Position

S INST	Depressing the CONT Switch will result in the execution of one instruction. This means that the machine state can be determined after each instruction. Examining and depositing into memory locations is a method of accomplishing this. The contents of the DATA DISPLAY LIGHTS are not necessarily meaningful.
S BUS CYCLE	For this mode to have any meaning, the Data Select Switch should be selecting the BUS REG (Bus Register). Depressing the CONT Switch will execute until the end of the next bus cycle. The Address Display Lights will then contain the address of the location that the bus cycle was performed at. (Virtual or Physical, depending on the position of the Address Select Switch). The DATA Display 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). Examine and deposit operations are not able to be used in this mode. Depressing the LOAD ARS, EXAM, or DEP Switch will not cause anything to happen. If an examine or deposit operation is desired, the S INST/ S BUS CYCLE Switch should be changed to select 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 examine or deposit.

11.8 FUNCTIONS OF SWITCHES & INDICATORS

11.8.1 Power Switch

OFF	Power to the processor is OFF.
POWER	Power to the processor is ON, and all console switches function normally.
LOCK	Power to the processor is ON, but the 7 control switches LOAD ADRS through START are disabled. All other switches are functional.

11.8.2 Control Switches

LOAD ADRS (Load Address)
When the LOAD ADRS Switch is depressed, the contents of the Switch Register are loaded into the Address Display. The address displayed in the Address Display Lights is a function of the position of the Address Select Switch.

EXAM (Examine)
Depressing the EXAM Switch causes the contents of the current location specified in the Address Display to be displayed in the DATA Display Register when the Data Select Switch is in the DATA PATHS position. The address in the Address Display will be mapped or unmapped depending on the position of the Address Select Switch. The location displayed in the Address Display Lights is also a function of that switch.

DEP (Deposit)
Raising the DEP Switch causes the current contents of the Switch Register to be deposited into the address specified by the current contents of the Address Display.

The address in the Address Display will be mapped or unmapped depending on the position of the Address Select Switch. The location displayed in the Address Display Lights is also a function of that switch.

CONT (Continue)
Depressing the CONT Switch causes the CPU to resume execution. The CONT Switch has no effect when the CPU is in RUN state.

ENABLE/HALT
The ENABLE/HALT Switch is a two position switch used to stop machine execution and to enable the system to run.

S/INST—S/BUS CYCLE (Single Instruction/Single Bus Cycle)
The S/INST—S/BUS CYCLE Switch affects only the operation of the CONTINUE Switch. It controls whether the machine stops after instructions or bus cycles. This switch has no effect on any switches when the ENABLE/HALT Switch is set to ENABLE.

START
The functions of the START Switch depend upon the setting of the ENABLE/HALT Switch as follows:

ENABLE	Starts execution
HALT	Clears the computer system

11.8.3 Switch Register
The switches are used to manually load data or an address into the processor, as determined by the control switches and the Address Select Switch.

Note that bits 0 to 15 of the current setting of the Switch Register may be read under program control from a read only register at address 17 777 570.

11.8.4 Lamp Test
The Lamp Test Switch (which is not labeled) is located between the Switch Register and the LOAD ADRS Switch. It is used for maintenance

purposes. When the Lamp Test Switch is raised, all console indicator lights should go on. An indicator which does not light is defective and should be replaced.

11.8.5 Address Select Switch

VIRTUAL (6 positions for User, Supervisor, & Kernel)	Uses a 16-bit Virtual Address where bits 16 to 21 are always OFF.
CONS PHY (Console Physical)	Uses a 22-bit Physical Address to perform console operations (e.g. LOAD ADRS, EXAM, & DEP).
PROG PHY (Program Physical)	Displays the 22-bit Physical Address of the current bus cycle that was generated by the Memory Management Unit.

11.8.6 Address Display

The ADDRESS Display lights are used to show the address of data being examined or just deposited. The address is interpreted as a Virtual or Physical Address as determined by the Address Select Switch.

11.8.7 Data Select Switch

DATA PATHS	The normal display mode, shows examined or deposited data.
BUS REG	The internal CPU register used for bus cycles.
μADRS FPP/CPU	The ROM address, FPP control micro-program (bits 15 to 8) and the CPU control micro-program (bits 7 to 0).
DISPLAY REGISTER	The contents of the Display Register. This has an address of 17 777 570.

11.8.8 Data Display

The Data Display lights are used to show the 16-bit word data just examined or deposited or other data within the CPU. The PARITY HIGH & LOW lights indicate the parity bit for the respective bytes on read operations; on write operations the bits are off. The interpretation of the data is determined by the Data Select Switch.

11.8.9 Status Indicator Lights

Error Indicators

PAR ERR	Lights to indicate a parity error during a reference to memory.
ADRS ERR	Lights to indicate any of the following addressing errors: a) Reference of non-existent memory b) Access control violation c) Reference of unassigned memory pages

Processor State

RUN	The CPU is executing program instructions. If the instruction being executed is a WAIT instruction, the RUN light will be on. The CPU will proceed from the WAIT on receipt of an external interrupt, or on console intervention.
PAUSE	The CPU is inactive because the current instruction execution has been completed as far as possible without more data from the UNIBUS or memory or the CPU is waiting to regain control of the UNIBUS (UNIBUS mastership).
MASTER	The CPU is in control of the UNIBUS (UNIBUS Master only when it needs the UNIBUS). The CPU relinquishes control of the UNIBUS during DMA and NPR data transfers

Mode

USER	The CPU is executing program instructions in USER mode.
SUPER (Supervisor)	The CPU is executing program instructions in SUPERVISOR mode.
KERNEL	The CPU is executing program instructions in KERNEL mode.
DATA	If on, the last memory reference was to D address space in the current CPU mode. If off, the last memory reference was to I address space in the current CPU mode.

Address

16 bit	Lights when the CPU is using 16-bit mapping.
18 bit	Lights when the CPU is using 18-bit mapping.
22 bit	Lights when the CPU is using 22-bit mapping.

11.9 M9301-YC BOOTSTRAP LOADER

FEATURES

- Contains bootstrap routines for a wide range of storage media
- Allows bootstrapping of any drive unit on a particular controller
- Runs diagnostic programs to test the basic CPU, Cache, and Main Memory
- Allows booting to selected physical memory segments in 32K increment
- Switch selectable default loading device

DESCRIPTION

The M9301-YC is a dedicated diagnostic bootstrap loader for use with the PDP-11/70. It contains a ROM organized as 512 16-bit words which are separated into hardware verification programs and bootstrap routines. It is a double height extended module which occupies rows E and F of slot one in the PDP-11/70 CPU.

DIAGNOSTICS

The diagnostic portion of the M9301-YC will test the basic CPU to include addressing modes, and most of the instructions available in the PDP-11/70. The ROM will then test memory from virtual addresses $100_{(8)}$ to $157776_{(8)}$. It does this first with the cache disabled to verify main memory, and then verifies the cache by retesting memory and enabling first one cache group, the other, and finally both cache groups simultaneously. Any errors detected will cause the program to halt. If any of the cache tests fail, the system can still be booted by pressing the console continue switch. The program will set the cache to force misses in both groups and proceed to boot.

The M9301-YC can be selected via the console switches <15:12> to test and load physical sections of memory other than the lowest 32K. The memory management and UNIBUS map can be set to use physical memory from 0 thru 512K Bytes. See Table 11-1.

TABLE 11-1 Bootstrap Option Selection (switch register settings)
The device codes are as follows:

Switch Register <03:06> Device Booted

1. TM11/TU10 MAGNETIC TAPE, TM11
2. TC11/TU56 DECTAPE, TC11-G
3. RK11/RK05 DECPACK DISK CARTRIDGE, RK11-D
4. RP11/RP03 DISK PACK, RP11-C
5. RESERVED FOR FUTURE DEVICE
6. RH70/TU16 MAGNETIC TAPE SYSTEM, TWU16
7. RH70/RP04 DISK PACK, RW04
10. RH70/RS04 FIXED HEAD DISK, RW04 (OR RWS03)
11. RX11/RX01 DISKETTE

The memory blocks are as follows:

Switch Register <08:11>

0. PHYSICAL MEMORY 00 000 000 – 00 077 776
1. PHYSICAL MEMORY 00 100 000 – 00 177 776

2. PHYSICAL MEMORY 00 200 000 – 00 277 776
3. PHYSICAL MEMORY 00 300 000 – 00 377 776
4. PHYSICAL MEMORY 00 400 000 – 00 477 776
5. PHYSICAL MEMORY 00 500 000 – 00 577 776
6. PHYSICAL MEMORY 00 600 000 – 00 677 776
7. PHYSICAL MEMORY 00 700 000 – 00 777 776
10. PHYSICAL MEMORY 01 000 000 – 01 077 776
11. PHYSICAL MEMORY 01 100 000 – 01 177 776
12. PHYSICAL MEMORY 01 200 000 – 01 277 776
13. PHYSICAL MEMORY 01 300 000 – 01 377 776
14. PHYSICAL MEMORY 01 400 000 – 01 477 776
15. PHYSICAL MEMORY 01 500 000 – 01 577 776
16. PHYSICAL MEMORY 01 600 000 – 01 677 776
17. PHYSICAL MEMORY 01 700 000 – 01 777 776

11/70 Bootstrap

The bootstrap portion of the program looks at the lower byte of the switch register to determine which one of 9 devices and which drive number to attempt the "BOOT" from, switches <02 : 00> select the drive number (0 – 7), and switches <06 : 03> select the device code (1 – 11). If the lower byte of the switch register is zero, the program will read the set of switches on the M9301-YC to determine the device and drive number. These switches can be set by field service to select a "DEFAULT BOOT" device.

If the bootstrap operation fails as a result of a hardware error in the peripheral device the program will do a "RESET" instruction and jump back to the test that sets up and turns on memory management and tests memory. Then the program will attempt to "BOOT" again.

STARTING PROCEDURE

To start operation of the M9301-YC, first set the console switch register to 17765000 and press Load Address. Then set the console switches for the desired memory section storage medium, and unit number (Table 11-1). With HALT switch in the ENABLE position, depress the START switch. This will cause the ROM diagnostic to run followed with a boot operation from the selected device. Failure of the diagnostic portion will be signified by a halt. Table 11-2 identifies the meaning of each error halt. If it is desired not to run the diagnostic portion of this sequence and to simply boot from the default device, the following procedure is used. First set the console switches to 17773000 and press Load Address. Place 0's in the switch register and with the HALT switch in the ENABLE position, press START. This will then cause the M9301-YC to read the switch setting located on the module to determine the device and unit number to boot from.

If it is desired only to boot from a device that is not the default device, a similar procedure is followed. First set the console switches to 17773000 and press Load Address. Then set the switch register to the desired memory section, storage medium, and unit number (Table 11-1) and with the HALT switch in the ENABLE position, press the START switch. This will cause the M9301-YC to boot the selected device.

Starting of the boot procedure can also be done under machine control. Execution of a jump instruction with the destination address of either

17765000 or 17773000, will cause the M9301-YC to sample the console switches and function as described above.

Table 11-2 Errors

List of error halts indexed by the address displayed

ADDRESS DISPLAYED	TEST NUMBER AND SUBSYSTEM UNDER TEST
17765004	TEST 1 BRANCH TEST
17765020	TEST 2 BRANCH TEST
17765036	TEST 3 BRANCH TEST
17765052	TEST 4 BRANCH TEST
17765066	TEST 5 BRANCH TEST
17765076	TEST 6 BRANCH TEST
17765134	TEST 7 REGISTER DATA PATH TEST
17765146	TEST 10 BRANCH TEST
17765166	TEST 11 CPU INSTRUCTION TEST
17765204	TEST 12 CPU INSTRUCTION TEST
17765214	TEST 13 CPU INSTRUCTION TEST
17765222	TEST 14 "COM" INSTRUCTION TEST
17765236	TEST 14 CPU INSTRUCTION TEST
17765260	TEST 15 CPU INSTRUCTION TEST
17765270	TEST 16 BRANCH TEST
17765312	TEST 16 CPU INSTRUCTION TEST
17765346	TEST 17 CPU INSTRUCTION TEST
17765360	TEST 20 CPU INSTRUCTION TEST
17765374	TEST 20 CPU INSTRUCTION TEST
17765450	TEST 21 KERNEL PAR TEST
17765474	TEST 22 KERNEL PDR TEST
17765510	TEST 23 JSR TEST
17765520	TEST 23 JSR TEST
17765530	TEST 23 RTS TEST
17765542	TEST 23 RTI TEST
17765550	TEST 23 JMP TEST
17765760	TEST 25 MAIN MEMORY DATA COMPARE ERROR
17766000	TEST 25 MAIN MEMORY PARITY ERROR (NO RECOVERY POSSIBLE FROM THIS ERROR)
17773644	TEST 26 CACHE MEMORY DATA COMPARE ERROR
17773654	TEST 26 CACHE MEMORY NO HIT PRESSING CONTINUE HERE WILL CAUSE BOOT ATTEMPT FORCING MISSES
17773736	TEST 27 CACHE MEMORY DATA COMPARE ERROR
17773746	TEST 27 CACHE MEMORY NO HIT PRESSING CONTINUE HERE WILL CAUSE BOOT ATTEMPT FORCING MISSES
17773764	TEST 25 OR 36 CACHE MEMORY PARITY ERROR PRESSING CONTINUE HERE WILL CAUSE BOOT ATTEMPT FORCING MISSES

ERROR RECOVERY

If the processor halts in one of the two Cache tests the error is recoverable. By pressing CONTINUE the program will either attempt to finish the test (if at either 17 773 644 or 17 773 736) or force MISSES in both groups of the Cache and attempt to boot the system monitor with the Cache fully disabled (if at either 17 773 654, 17 773 746, 17 773 764). The run time for this program is approximately 3 seconds.

APPENDIX A

UNIBUS ADDRESSES

A.1 INTERRUPT & TRAP VECTORS

000	(reserved)
004	CPU errors
010	Illegal & reserved instructions
014	BPT, breakpoint trap
020	IOT, input/output trap
024	Power Fail
030	EMT, emulator trap
034	TRAP instruction
040	System software
044	System software
050	System software
054	System software
060	Console Terminal, keyboard/reader
064	Console Terminal, printer/punch
070	PC11, paper tape reader
074	PC11, paper tape punch
100	KW11-L, line clock
104	KW11-P, programmable clock
110	
114	Memory system errors
120	XY Plotter
124	DR11-B DMA interface; (DA11-B)
130	AD01, A/D subsystem
134	AFC11, analog subsystem
140	AA11, display
144	AA11, light pen
150	
154	
160	
164	
170	User reserved
174	User reserved
200	LP11/LS11, line printer
204	RS04/RF11, fixed head disk
210	RC11, disk
214	TC11, DECtape
220	RK11, disk
224	TU16/TM11, magnetic tape
230	CD11/CM11/CR11, card reader
234	UDC11, digital control subsystem
240	PIRQ, Program Interrupt Request (11/45)

244	Floating Point Error
250	Memory Management
254	RP04/RP11 disk pack
260	TA11, cassette
264	
270	User reserved
274	User reserved
300	(start of floating vectors)

A.2 FLOATING VECTORS

There is a floating vector convention used for communications (and other) devices that interface with the PDP-11. These vector addresses are assigned in order starting at 300 and proceeding upwards to 777. The following Table shows the assigned sequence. It can be seen that the first vector address, 300, is assigned to the first DC11 in the system. If another DC11 is used, it would then be assigned vector address 310, etc. When the vector addresses have been assigned for all the DC11's (up to a maximum of 32), addresses are then assigned consecutively to each unit of the next highest-ranked device (KL11 or DP11 or DM11, etc.), then to the other devices in accordance with the priority ranking.

Priority Ranking for Floating Vectors

(starting at 300 and proceeding upwards)

Rank	Device	Vector Size (in octal)	Max No.
1	DC11	$(10)_9$	32
2	KL11, DL11-A, DL11-B	10	16
3	DP11	10	32
4	DM11-A	10	16
5	DN11	4	16
6	DM11-BB	4	16
7	DR11-A	10*	32
8	DR11-C	10*	32
9	PA611 Reader	4*	16
10	PA611 Punch	4*	16
11	DT11	10*	8
12	DX11	10*	4
13	DL11-C, DL11-D, DL11-E	10	31
14	DJ11	10	16
15	DH11	10	16
16	GT40	10	1
17	LPS11	30*	1
18	DQ11	10	16
19	KW11-W	10	1
20	DU11	10	16

*—The first vector for the first device of this type must always be on a $(10)_8$ boundary.

A.3 FLOATING ADDRESSES

There is a floating address convention used for communications (and other) devices interfacing with the PDP-11. These addresses are assigned in order starting at 760 010 and proceeding upwards to 763 776.

Floating addresses are assigned in the following sequence:

Rank	Device	First Address (if only floating address device in the system)
1	DJ11	760 010
2	DH11	760 020
3	DQ11	760 030
4	DU11	760 040

A.4 DEVICE ADDRESSES

Address	Register	
777 776	Processor Status word (PS)	
777 774	Stack Limit (SL)	
777 772	Program Interrupt Request (PIR)	
777 770	Microprogram Break	
777 766	CPU Error	
777 764	System I/D	
777 762	Upper Size	System Size
777 760	Lower Size	System Size
777 756		
777 754		
777 752	Hit/Miss	
777 750	Maintenance	
777 746	Control	
777 744	Memory System Error	
777 742	High Error Address	
777 740	Low Error Address	
777 717	User	R6 (SP)
777 716	Supervisor	R6 (SP)
777 715	General registers, Set 1	R5
777 714	General registers, Set 1	R4
777 713	General registers, Set 1	R3
777 712	General registers, Set 1	R2
777 711	General registers, Set 1	R1
777 710	General registers, Set 1	R0
777 707		R7 (PC)
777 706	Kernel	R6 (SP)
777 705	General registers, Set 0	R5
777 704	General registers, Set 0	R4
777 703	General registers, Set 0	R3
777 702	General registers, Set 0	R2
777 701	General registers, Set 0	R1
777 700	General registers, Set 0	R0

Address	Device	Register
777 676 – 777 660	User Data PAR, reg 0-7	
777 656 – 777 640	User Instruction PAR, reg 0-7	
777 636 – 777 620	User Data PDR, reg 0-7	
777 616 – 777 600	User Instruction PDR, reg 0-7	
777 576		(MMR2)
777 574	Memory Mgt regs,	(MMR1)
777 572		(MMR0)
777 570	Console Switch & Display Register	
777 566		printer/punch data
777 564	Console Terminal,	printer/punch status
777 562		keyboard/reader data
777 560		keyboard/reader status
777 556		punch data (PPB)
777 554	PC11/PR11,	punch status (PPS)
777 552		reader data (PRB)
777 550		reader status (PRS)
777 546	KW11-L, clock status (LKS)	
777 516		printer data
777 514	LP11/LS11/LV11,	printer status
777 512		
777 510		
777 506		
777 504		
777 502	TA11,	cassette data (TADB)
777 500		cassette status (TACS)
777 476		look ahead (ADS)
777 474		maintenance (MA)
777 472		disk data (DBR)
777 470	RF11,	adrs ext error (DAE)
777 466		disk address (DAR)
777 464		current mem adrs (CMA)
777 462		word count (WC)
777 460		disk status (DCS)
777 456		disk data (RCDB)
777 454		maintenance (RCMN)
777 452		current address (RCCA)
777 450	RC11,	word count (RCWC)
777 446		disk status (RCCS)
777 444		error status (RCER)
777 442		disk address (RCDA)
777 440		look ahead (RCLA)

Address	Device	Register
777 436		#8
777 434		#7
777 432		#6
777 430	DT11, bus switch	#5
777 426		#4
777 424		#3
777 422		#2
777 420		#1

Address	Device	Register
777 416		disk data (RKDB)
777 414		maintenance
777 412		disk address (RKDA)
777 410	RK11,	bus address (RKBA)
777 406		word count (RKWC)
777 404		disk status (RKCS)
777 402		errorr (RKER)
777 400		drive status (RKDS)

Address	Device	Register
777 356		
777 354		
777 352		
777 350		DECtape data (TCDT)
777 346	TC11,	bus address (TCBA)
777 344		word count (TCWC)
777 342		command (TCCM)
777 340		DECtape status (TCST)

Address	Device	Register
777 336	KE11-A, EAE #2	
777 320		

Address	Device	Register
777 316		arithmetic shift
777 314		logical shift
777 312		normalize
777 310	KE11-A, EAE #1,	step count/status register
777 306		multiply
777 304		multiplier quotient
777 302		accumulator
777 300		divide

Address	Device	Register	Device	Register
777 166				data (CDDB)
777 164	CR11/	data (CRB2) comp		cur adrs (CDBA)
777 162	CM11,	data (CRB1)	CD11,	col count (CDCC)
777 160		status (CRS)		status (CDST)

Address	Device	Register
776 776		
776 774		
776 772	AD01,	A/D data (ADDB)
776 770		A/D status (ADCS)

Address	Device	Register
776 766		register 4 (DAC4)
776 764		register 3 (DAC3)
776 762		register 2 (DAC2)
776 760	AA11 #1,	register 1 (DAC1)
776 756		D/A status (CSR)
776 754		

Address	RP04	RP11
776 752	cont & status #3 (RPCS3)	
776 750	bus adrs ext (RPBAE)	
776 746	ECC pattern (RPEC2)	
776 744	ECC position (RPEC1)	
776 742	error #3 (RPER3)	
776 740	error #2 (RPER2)	
776 736	cur cylinder (RPCC)	silo memory (SILO)
776 734	desired cyl (RPDC)	cyl adrs (SUCA)
776 732	offset (RPOF)	maint 3 (RPM3)
776 730	serial number (RPSN)	maint 2 (RPM2)
776 726	drive type (RPDT)	maint 1 (RPM1)
776 724	maintenance (RPMR)	disk adrs (RPDA)
776 722	data buffer (RPDB)	cyl adrs (RPCA)
776 720	RP04, look ahead (RPLA)	RP11, bus adrs (RPBA)
776 716	attn summary (RPAS)	word count (RPWC)
776 714	error #1 (RPER1)	disk status (RPCS)
776 712	drive status (RPDS)	error (RPER)
776 710	cont & status #2 (RPCS2)	disk status (RPDS)
776 706	sector/track adrs (RPDA)	
776 704	UNIBUS address (RPBA)	
776 702	word count (RPWC)	
776 700	cont & status #1 (RPCS1)	

Address	Device	Unit
776 676	DL11-A, -B,	#16
776 500		#1
776 476	AA11,	#5
776 400		#2
776 376	DX11	
776 200		
776 176	DL11-C, -D, -E,	#31
775 610		#1
775 576	DS11,	#4
775 400		#1
775 376	DN11,	#16
775 200		#1
775 176	DM11,	#16
775 000		#1

Address	Device	Register
774 776		#1
	DP11,	
774 400		#32
774 376		#32
	DC11,	
774 000		#1
773 766		
	PDP-11/70 diagnostic bootstrap (half of it)	
773 000		
772 776		
	PA611 typeset punch	
772 700		
772-676		
	PA611 typeset reader	
772 600		
772 576		maintenance (AFMR)
772 574	AFC11,	MX channel/gain (AFCG)
772 572		flying cap data (AFBR)
772 570		flying cap status (AFCS)
772 556		
	XY11 plotter	
772 550		
772 546		
772 544		counter
772 542	KW11-P,	count set
772 540		clock status
772 536		
772 534		
772 532		read lines (MTRD)
772 530		tape data (MTD)
772 526	TM11,	memory address (MTCMA)
772 524		byte record counter (MTBRC)
772 522		command (MTC)
772 500		tape status (MTS)
772 516	Memory Mgt reg (MMR3)	
772 476		cont & status #3 (MTCS3)
772 474		bus adrs ext (MTBAE)
772 472		tape control (MTTC)
772 470		serial number (MTSN)
772 466		drive type (MTDT)
772 464		maintenance (MTMR)
772 462		data buffer (MTDB)
772 460		check character (MTCK)
772 456	TU16,	attention summary (MTAS)
772 454		error (MTER)
772 452		drive status (MTDS)
772 450		cont & status #2 (MTCS2)

Address		Description
772 446		frame count (MTFC)
772 444		UNIBUS address (MTBA)
772 442		word count (MTWC)
772 440		cont & status #1 (MTCS1)
772 436 – 772 430	DR11-B #2	
772 416		data (DRDB)
772 414	DR11-B #1,	status (DRST)
772 412		bus address (DRBA)
772 410		word count (DRWC)
772 376 – 772 360	Kernel Data PAR, reg 0-7	
772 356 – 772 340	Kernel Instruction PAR, reg 0-7	
772 336 – 772 320	Kernel Data PDR, reg 0-7	
772 316 – 772 300	Kernel Instruction PDR, reg 0-7	
772 276 – 772 260	Supervisor Data PAR, reg 0-7	
772 256 – 772 240	Supervisor Instruction PAR, reg 0-7	
772 236 – 772 220	Supervisor Data Descriptor PDR, reg 0-7	
772 216 – 772 200	Supervisor Instruction Descriptor PDR, reg 0-7	
772 136 – 772 110	UNIBUS Memory Parity	
772 072		cont & status #3 (RSCS3)
772 070		bus adrs ext (RSBAE)
772 066		drive type (RSDT)
772 064		maintenance (RSMR)
772 062		data buffer (RSDB)
772 060		look ahead (RSLA)
772 056		attention summary (RSAS)
772 054	RS04,	error (RSER)

Address	Device	Register
772 052		drive status (RSDS)
772 050		control & status #2 (RSCS2)
772 046	RS04,	desired disk adrs (RSDA)
772 044		UNIBUS address (RSBA)
772 042		word count (RSWC)
772 040		control & status #1 (RSCS1)
772 016 – 772 010	GT40 #2	
772 006		Y axis
772 004		X axis
772 002	GT40 #1	status
772 000		program counter
771 776		status (UDCS)
771 774	UDC11,	scan (UDSR)
771 772		
771 770		
771 776 – 771 000	UDC functional I/O modules	
770 776 – 770 700	KG11,	#8 – #1
770 676 – 770 500	DM11-BB,	#16 – #1
770 436		DMA
770 434		
770 432		
770 430		
770 426		
770 424		
770 422		ext DAC
770 420		D/A YR
770 416		D/A XR
770 414		D/A SR
770 412	LPS11,	D I/O output
770 410		D I/O input
770 406		CKBR
770 404		CKSR
770 402		ADBR
770 400		ADSR
770 366 – 770 200	UNIBUS Map	
767 776 – 766 000	GT40 bootstrap	

765 776 765 000	PDP-11/70 diagnostic bootstrap (half of it)
763 776	(top of floating addresses)
760 010	(start of floating addresses)

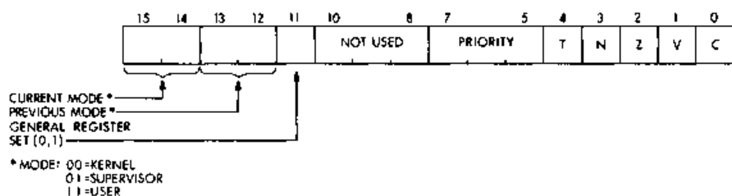
NOTE

For the PDP-11/70, all addresses in Appendix A between 777 777 and 776 000 should be prefixed with 17. The address range is then 17 777 777 to 17 760 000.

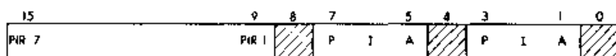
APPENDIX B

CPU & MASS STORAGE DEVICE REGISTERS

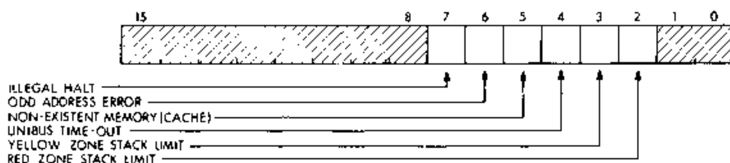
Processor Status Word (PS) 17 777 776



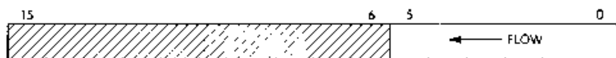
Program Interrupt Request (PIR) 17 777 772



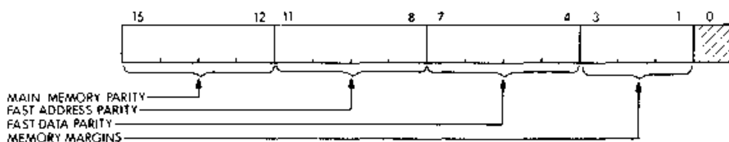
CPU Error Register 17 777 766



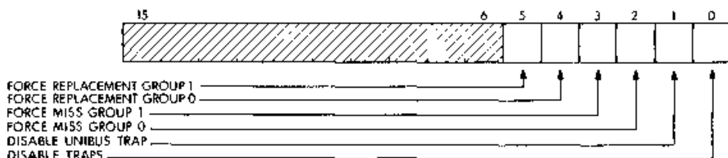
Hit/Miss Register 17 777 752



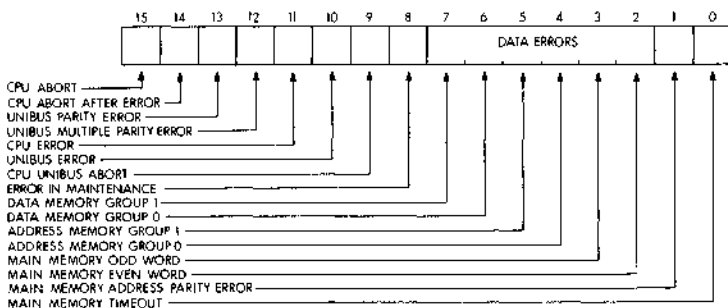
Maintenance Register 17 777 750



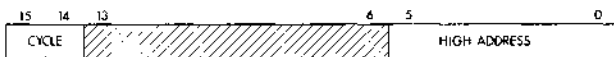
Control Register 17 777 746



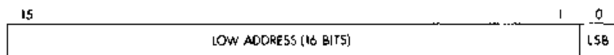
Memory System Error Register 17 777 744



High Error Address Register 17 777 742



Low Error Address Register 17 777 740



RP04 Registers

Register	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RPCS1 (776700)	SC	TRE	MCPE	Ø	DVA	PSEL	A17	A16	RDY	IE	F4	F3	F2	F1	FØ	GO
RPWC (776702)	WC 15	WC 14	WC 13	WC 12	WC 11	WC 10	WC 9	WC 8	WC 7	WC 6	WC 5	WC 4	WC 3	WC 2	WC 1	WC Ø
RPBA (776704)	BA 15	BA 14	BA 13	BA 12	BA 11	BA 10	BA 9	BA 8	BA 7	BA 6	BA 5	BA 4	BA 3	BA 2	BA 1	BA Ø
RPDA (776706)	Ø	Ø	Ø	TA 16	TA 8	TA 4	TA 2	TA 1	Ø	Ø	Ø	SA 16	SA 8	SA 4	SA 2	SA 1
RPCS2 (776710)	DLT	WCE	UPE	NED	NEM	PGE	MXF	MDPE	OR	IR	CLR	PAT	BAI	U2	U1	UØ
RPDS (776712)	ATA	ERR	PIP	MOL	WRL	LST	PGM	DPR	DRY	VV	DE1	DL64	GRV	DIGB	DF20	DF5
RPER1 (776714)	DCK	UNS	OPI	DTE	WLE	IAE	AOE	HCRC	HCE	ECH	WCF	FER	PAR	RMR	ILR	ILF
RPAS (776716)	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	ATA 7	ATA 6	ATA 5	ATA 4	ATA 3	ATA 2	ATA 1	ATA Ø
RPLA (776720)	Ø	Ø	Ø	Ø	Ø	SC 4	SC 3	SC 2	SC 1	SC Ø	EXT 1	EXT Ø	Ø	Ø	Ø	Ø
RPDB (776722)	DB 15	DB 14	DB 13	DB 12	DB 11	DB 10	DB 9	DB 8	DB 7	DB 6	DB 5	DB 4	DB 3	DB 2	DB 1	DB Ø
RPMR (776724)	Ø	Ø	Ø	Ø	Ø	Ø	SBD	ZDI	DEN	ECCE	MWR	MRD	MSCLK	MIND	MCLK	DMD
RPDT (776726)	NBA	TAP	MOH	Ø	DRQ	Ø	Ø	DT 8	DT 7	DT 6	DT 5	DT 4	DT 3	DT 2	DT 1	DT Ø
RPSN (776730)	SN 38	SN 34	SN 32	SN 31	SN 28	SN 24	SN 22	SN 21	SN 18	SN 14	SN 12	SN 11	SN 8	SN 4	SN 2	SN 1
RPOF (776732)	SGCH	Ø	Ø	FMT 22	ECCI	HCI	Ø	Ø	OFS 7	OFS 6	OFS 5	OFS 4	OFS 3	OFS 2	OFS 1	OFS Ø
RPDC (776734)	Ø	Ø	Ø	Ø	Ø	Ø	Ø	DC 256	DC 128	DC 64	DC 32	DC 16	DC 8	DC 4	DC 2	DC 1
RPCC (776736)	Ø	Ø	Ø	Ø	Ø	Ø	Ø	CC 256	CC 128	CC 64	CC 32	CC 16	CC 8	CC 4	CC 2	CC 1
RPER2 (776740)	AC UNS	Ø	PLU	30VU	IXE	NAS	MHS	WRU	FEN	TUF	TDF	MSE	CSU	WSU	CSF	WCU
RPER3 (776742)	OCYL	SKI	Ø	Ø	Ø	Ø	Ø	Ø	Ø	ACL	DCL	PRE	UWR	Ø	VUF	PSU
RPEC1 (776744)	Ø	Ø	Ø	BLC 4096	BLC 2048	BLC 1024	BLC 512	BLC 256	BLC 128	BLC 64	BLC 32	BLC 16	BLC 8	BLC 4	BLC 2	BLC 1
RPEC2 (776746)	Ø	Ø	Ø	Ø	BIT 11	BIT 10	BIT 9	BIT 8	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT Ø
RPBAE (776750)	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	BA 21	BA 20	BA 19	BA 18	BA 17	BA 16
RPCS3 (776752)	APE	DPE OW	DPE EW	WCE OW	WCE EW	DBL	Ø	Ø	Ø	IE	Ø	Ø	IPCK 3	IPCK 2	IPCK 1	IPCK Ø

RS04/RS03 Registers

Register	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RSCS1-772040	SC	TRE	MCPE	0	DVA	PSEL	A17	A16	RDY	IE	F4	F3	F2	F1	F0	GO
RSWC-772042	WC 15	WC 14	WC 13	WC 12	WC 11	WC 10	WC 09	WC 08	WC 07	WC 06	WC 05	WC 04	WC 03	WC 02	WC 01	WC 00
RSBA-772044	BA 15	BA 14	BA 13	BA 12	BA 11	BA 10	BA 09	BA 08	BA 07	BA 06	BA 05	BA 04	BA 03	BA 02	BA 01	0
RSDA-772046	SP 03	SP 02	SP 01	SP 00	TA 05	TA 04	TA 03	TA 02	TA 01	TA 00	SA 05	SA 04	SA 03	SA 02	SA 01	SA 00
RSCS2-772050	DLT	WCE	UPE	NED	NEM	PGE	MXF	MDPE	OR	IR	CLR	PAT	BAI	U2	U1	U0
RSDS-772052	ATA	ERR	PIP	MOL	WRL	LBT	0	DPR	DRY	0	0	0	0	0	0	0
RSER-772054	DCK	UNS	OPI	DTE	WLE	IAE	AO	0	0	0	0	0	PAR	RMR	ILR	ILF
RSAS-772056	0	0	0	0	0	0	0	0	ATA 07	ATA 06	ATA 05	ATA 04	ATA 03	ATA 02	ATA 01	ATA 00
RSLA-772060	0	0	0	0	CS 05	CS 04	CS 03	CS 02	CS 01	CS 00	SF 05	SF 04	SF 03	SF 02	SF 01	SF 00
RSDB-772062	DB 15	DB 14	DB 13	DB 12	DB 11	DB 10	DB 09	DB 08	DB 07	DB 06	DB 05	DB 04	DB 03	DB 02	DB 01	DB 00
RSMR-772064	RWDLK	MWDT	CRCW	MWDB	SB	LSR	AC	SP	WRT	RD	MRDT	MLNC	MCLK	MRDB	0	DMD
RSDT-772066	NSA	TAP	MOH	7CH	DRQ	SPR	0	DT 08	DT 07	DT 06	DT 05	DT 04	DT 03	DT 02	DT 01	DT 00
RSBAE-772070	0	0	0	0	0	0	0	0	0	0	BA 21	BA 20	BA 19	BA 18	BA 17	BA 16
RSCS3-772072	APE	DPE OW	DPE EW	WCE OW	WCE EW	DBL	0	0	0	IE	0	0	IPCK 3	IPCK 2	IPCK 1	IPCK 0

TU16 Registers

Register	Address	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MTCS1	772440	SC	TRE	MCPE	O	DVA	PSEL	A17	A16	RDY	IE	F4	F3	F2	F1	FØ	GO
MTWC	772442	WC 15	WC 14	WC 13	WC 12	WC 11	WC 10	WC 09	WC 08	WC 07	WC 06	WC 05	WC 04	WC 03	WC 02	WC 01	WC ØØ
MTBA	772444	BA 15	BA 14	BA 13	BA 12	BA 11	BA 10	BA 09	BA 08	BA 07	BA 06	BA 05	BA 04	BA 03	BA 02	BA 01	BA ØØ
MTFC	772446	FC 15	FC 14	FC 13	FC 12	FC 11	FC 10	FC 09	FC 08	FC 07	FC 06	FC 05	FC 04	FC 03	FC 02	FC 01	FC ØØ
MTCS2	772450	DLT	WCE	UPE	NED	NEM	PGE	MXF	MDPE	OR	IR	CLR	PAT	BAI	U2	U1	UØ
MTDS	772452	ATA	ERR	PIP	MOL	WRL	EOT	Ø	DPR	DRY	SSC	PES	SDWN	IOB	TM	BOT	SLA
MTER	772454	COR/ CRC	UNS	OPI	DTE	NEF	CS/ ITM	FCE	NSG	PEF/ LRC	INC/ VPE	DPAR	FMT	CPAR	RMR	ILR	ILF
MTAS	772456	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	ATA Ø7	ATA Ø6	ATA Ø5	ATA Ø4	ATA Ø3	ATA Ø2	ATA Ø1	ATA ØØ
MTCK	772460	Ø	Ø	Ø	Ø	Ø	Ø	Ø	CRCP / DTP	CRC7 / DT7	CRC6 / DT6	CRC5 / DT5	CRC4 / DT4	CRC3 / DT3	CRC2 / DT2	CRC1 / DT1	CRCØ / DTØ
MTDB	772462	DB 15	DB 14	DB 13	DB 12	DB 11	DB 10	DB 09	DB 08	DB 07	DB 06	DB 05	DB 04	DB 03	DB 02	DB 01	DB ØØ
MTMR	772464	MDF Ø8	MDF Ø7	MDF Ø6	MDF Ø5	MDF Ø4	MDF Ø3	MDF Ø2	MDF Ø1	MDF ØØ	200 BPI CLK	MC	MOP Ø3	MOP Ø2	MOP Ø1	MOP ØØ	MM
MTDT	772466	NSA	TAP	MOH	7CH	DRØ	SPR	Ø	DT Ø8	DT Ø7	DT Ø6	DT Ø5	DT Ø4	DT Ø3	DT Ø2	DT Ø1	DT ØØ
MTSN	772470	SN 15	SN 14	SN 13	SN 12	SN 11	SN 10	SN 09	SN 08	SN 07	SN 06	SN 05	SN 04	SN 03	SN 02	SN 01	SN ØØ
MTTC	772472	ACCL	TCW	FCS	EAO DTE	Ø	DEN Ø2	DEN Ø1	DEN ØØ	FMT SEL Ø3	FMT SEL Ø2	FMT SEL Ø1	FMT SEL ØØ	EV PAR	SS 2	SS 1	SS Ø
MTBAE	772474	O	O	O	O	O	O	O	O	O	O	BA 21	BA 20	BA 19	BA 18	BA 17	BA 16
MTCS3	772476	APE	DPE OW	DPE EW	DPE OW	DPE EW	DBL	O	O	O	IE	O	O	IPCK 3	IPCK 2	IPCK 1	IPCK 0

APPENDIX C

INSTRUCTION TIMING

C.1 INSTRUCTION EXECUTION TIME

The execution time for an instruction depends on the instruction itself, the modes of addressing used, and the type of memory being referenced. In the most general case, the Instruction Execution Time is the sum of a Source Address Time, and an Execute, Fetch Time.

Instr Time = SRC Time + DST Time + EF Time

Some of the instructions require only some of these times, and are so noted. Times are typical; processor timing, with core memory, may vary +15% to −10%.

C.1.1 BASIC INSTRUCTION SET TIMING

Double Operand

all instructions,
except MOV: Instr Time = SRC Time + DST Time
(but including MOVB) + EF Time
MOV Instruction: Instr Time = SRC Time + EF Time
(word only)

Single Operand
all instructions: Instr Time = DST Time + EF Time or
Instr Time = SRC Time + EF Time

Branch, Jump, Control, Trap & Misc
all instructions: Instr Time = EF Time

C.1.2 USING THE CHART TIMES

To compute a particular instruction time, first find the instruction "EF" Time. Select the proper EF Time for the SRC and DST modes. Observe all "NOTES" to the EF Time by adding the correct amount to basic EF number.

Next, note whether the particular instruction requires the inclusion of SRC and DST Times, if so, add the appropriate amounts to correct EF number.

C.1.3 CHART TIMES

The times given in the chart for Cache "hits"; that is, all the read cycles are assumed to be in the Cache. The number of read cycles in each subset of the instruction is also included so that timing can be calculated for a specific case of hits and misses, or timing can be calculated based on an average hit rate.

a) Specific hits and misses
Add 1.02 μsec for each read cycle which is a miss instead of a hit.

b) Average hit rate
If P_H is the percent of reads that are hits, add $1.02 \times (1 - P_H) \times$ (Number of read cycles) to the instruction timing.

For example, an ADD A,B instruction using Mode 6 (indexed) address modes:

1) All Hits:

SRC time	= 0.60 μsec	2 read cycles
DST time	= 0.60 μsec	2 read cycles
EF time	= 1.35 μsec	1 read cycle
TOTAL	= 2.55 μsec	5 read cycles

2) 4 Hits, 1 Miss
Total = 2.55 + 1.02
= 3.57 μsec

3) Read hit rate of 90%
Total = 2.55 + (1.02) (.1) (5)
= 3.06 μsec

C.1.4 NOTES

1. The times specified generally apply to Word instructions. In most cases Even Byte instructions have the same time, with some Odd Byte instructions taking longer. All exceptions are noted.
2. Timing is given without regard for NRP or BR serving. Core memory is assumed to be located within the first 128K memory unit.
3. Times are not affected if Memory Management is enabled.
4. All times are in microseconds.

C.1.5 SOURCE ADDRESS TIME

Instruction	Source Mode	SRC Time	Read Memory Cycles
Double Operand	0	.00	0
	1	.30	1
	2	.30	1
	3	.75	2
	4	.45	1
	5	.90	2
	6	.60	2
	7	1.05	3

C.1.6 DESTINATION ADDRESS TIME

Instruction	DST Mode	DST Time (A)	Read Memory Cycles
Single Operand and Double Operand (except MOV, MTPI, MTPD, JMP, JRS	0	.00	0
	1	.30	1
	2	.30	1
	3	.75	2
	4	.45	1
	5	.90	2
	6	.60	2
	7	1.05	3

NOTE (A): Add .15 μsec for odd byte instructions, except DST Mode 0.

C.1.7 EXECUTE, FETCH TIME

Double Operand

Instruction (Use with SRC Time and DST Time)	EF Time (SRC Mode 0) (DST Mode 0)	Read Mem Cyc	EF Time (SRC Mode 1-7) (DST Mode 0)	Read Mem Cyc	EF Time (SRC Mode 0-7) (DST Mode 1-7)	Read Mem Cyc
ADD, SUB, BIC, BIS MOVB	.30 (D)	1	.45 (D)	2	1.20 (C)	1
CMP, BIT	.30 (D)	1	.45 (D)	1	.45 (C)	1
XOR	.30 (D)	1	.30 (D)	1	1.20	1

NOTE (C): Add 0.15 μsec if SRC is R1 to R7 and DST is R6 or R7.
NOTE (D): Add 0.3 μsec if DST is R7.

Instruction (Use with SRC Time)	DST Mode	DST Register	EF Time (SRC Mode = 0)	EF Time (SRC Mode = 1-7)	Read Memory Cycles
MOV	0	0-6	.30	.45	1
	0	7	.60	.75	1
	1	0-7	1.20	1.20	1
	2	0-7	1.20	1.20	1
	3	0-7	1.65	1.65	2
	4	0-7	1.35	1.35	1
	5	0-7	1.80	1.80	2
	6	0-7	1.50	1.65	2
	7	0-7	1.95	2.10	3

Single Operand

Instruction (Use with DST Time)	EF TIME (DST Mode = 0)	Memory Cycles	EF Time (DST Mode 1 to 7)	Read Memory Cycles
CLR, COM, INC, DEC, ADC, SBC, ROL, ASL, SWAB, SXT	.30 (J)	1	1.20	1
NEG	.75	1	1.50	1
TST	.30 (J)	1	.45	1
ROR, ASR	.30 (J)	1	1.20 (H)	1
ASH, ASHC	.75 (I)	1	.90 (I)	1

NOTE (H): Add 0.15 μsec if odd byte.
NOTE (I): Add 0.15 μsec per shift.
NOTE (J): Add 0.30 μsec if DST is R7.

Instruction (Use with SRC Times)	EF Time	Read Memory Cycles
MUL	3.30	1
DIV		
by zero	.90	1
shortest	7.05	1
longest	8.55	1

Instruction	EF Time	Read Memory Cycles	
MFPI	1.50	1	use with SRC times
MFPD	1.50	1	

Instruction	DST Mode	Instruction Time	Read Memory Cycles
MTPI	0	.90	1
MTPD	1	1.65	2
	2	1.65	2
	3	2.10	3
	4	1.80	2
	5	2.25	3
	6	2.10	3
	7	2.55	4

Branch Instructions

Instruction	Instr Time (Branch)	Instr Time (No Branch)	Read Memory Cycles
BR, BNE, BEQ, BPL, BMI, BVC, BVS, BCC, BCS, BGE, BLT, BGT, BLE, BHI, BLOS, BHIS, BLO	.60	.30	1
SOB	.60	.75	1

Jump Instructions

Instruction	DST Mode	Instr Time	Read Memory Cycles
	1	.90	1
	2	.90	1
	3	1.20	2
JMP	4	.90	1
	5	1.35	2
	6	1.05	2
	7	1.50	3
	1	1.95	1
	2	1.95	1
	3	2.25	2
JSR	4	1.95	1
	5	2.40	2
	6	2.10	2
	7	2.55	3

Control, Trap & Miscellaneous Instructions

Instruction	Instr Time	Read Memory Cycles
RTS	1.05	2
MARK	.90	2
RTI, RTT	1.50	3
SET N, Z, V, C CLR, N, Z, V, C	.60	1
HALT	1.05	0
WAIT WAIT Loop for a BR is .3 μsec.	.45	0
RESET	10ms	1
IOT, EMT, TRAP, BRT	3.30	3
SPL	.60	1
INTERRUPT First Device	2.31	2

C.1.8 EFFECTIVE MEMORY CYCLE TIME

The overall effective cycle time of the CPU can be calculated from the following formula:

$$TC_E = P_R \times [(P_H \times TC_H) + (1 - P_H) TC_M] + (1 - P_R) TC_W$$

Where TC_E = Effective cycle time
TC_H = Cycle time for a read hit = 0.30 μsec
TC_M = Cycle time for a read miss = 1.32 μsec
TC_W = Cycle time for a write = 0.75 μsec
P_R = Percent of cycles that are reads
P_H = Percent of reads that are hits

Thus, for an average PDP-11/70 program which has a read rate of 91% and a read hit rate of 93%, the effective cycle time is:

$$TC_E = .91 \times [(.93 \times .30) + (.07 \times 1.32)] + (.09 \times .75) = .41 \mu sec$$

C.2 FLOATING POINT INSTRUCTION TIMING

INTRODUCTION

Floating Point instruction times are calculated in a manner similar to the calculation of CPU instruction timing. Due to the fact that the FP11-C is a separate processor operating in parallel with the main processor however, the calculation of Floating Point instruction times must take this parallel processing or overlap into account. The following is a description of the method used to calculate the effective Floating Point instruction execution times.

DEFINITIONS

Preinteraction — CPU time required to decode a Floating Point instruction OP Code and to store the general register referred to in the Floating Point instruction in a temporary Floating Point register (FPR). This time is fixed at 450 ns.

Address Calculation — CPU time required to calculate the address of the operand. This time is dependent on the addressing mode specified. Refer to Table C-1.

Wait Time — CPU time spent waiting for completion by the Floating Point Processor of a previous Floating Point instruction in the case of Load Class instructions. For Store Class instructions, the Wait Time is the summation of time during which the Floating Point completes a previous Floating Point instruction and Floating Point execution time for the store class instruction. Wait Time is calculated as follows:

Load Class Instructions:

Wait Time = [Floating Point execution time (previous FP instruction)] – [Disengage and Fetch Time (previous FP instruction)] – [CPU execution time for interposing nonfloating point instruction] – [Preinteraction time] – [Address Calculation Time]. If the result is ≤ 0 the Wait Time is 0.

Store Class Instructions:

Wait Time = {[Floating Point execution time (previous Floating Point instruction)] – [CPU execution time for interposing nonFP instruction] – Disengage and Fetch time (previous FP instruction)] – [Preinteraction]}* + Floating Point execution time] – [Address Calculation time]. If Wait Time calculation result is ≤ 0 the Wait Time is 0.

* If result of calculation in { } is ≤ 0 then it becomes 0.

Resync Time	If the CPU must wait for the Floating Point Processor (i.e., Wait Time = 0), an additional 450 ns must be added to the Effective Execution time of the instruction. If Wait Time ≠ 0 then Resync Time = 0.
Interaction Time	CPU time required to actually initiate Floating Point Processor operation.
Argument Transfer Time	CPU time required to fetch and transfer to the Floating Point Processor the required operand. This time is 300 ns × the number of 16-bit words read from Memory (Load Class Floating Point Instructions), or 1200 ns × the number of 16-bit words written to Memory (Store Class Instructions).
Disengage and Fetch Time	CPU time required to fetch the next instruction from Memory. This time is 300 ns.
Floating Point Execution Time	Time required by the Floating Point Processor to complete a Floating Point instruction once it has received all arguments (Load Class Instructions). Execution times are contained in Table C-2.
Effective Execution Time	Total CPU time required to execute a Floating Point instruction.

Effective Execution Time = Preinteraction + Address Calculation + Wait Time + Resync Time + Interaction Time + Argument Transfer + Disengage and Fetch.

Table C-1 Address Calculation Times

Mode	Address Calculation Time
0	0 nsec
1	300
2	300
3	600
4	300
5	750
6	600
7	1050

Table C-2 FP11-C Execution Times

	MIN	MAX	TYP
LDF	360 nsec	360 nsec	
LDD	360	360	
ADDF	900	2520	950
ADDD	900	4140	980

Table C-2 FP11-C Execution Times (Cont.)

	MIN	MAX	TYP
SUBF	900	1980	1130
SUBD	900	4140	1160
MULF	1800	3440	2520
MULD	3060	6220	4680
DIVF	1920	6720	3540
DIVD	3120	14400	6000
MODF	2880	5990	
MODD	3780	9770	
LDCFD	420	420	
LDCDF	540	540	
STF*	0		
STD*	0		
CMPF	540	1080	
CMPD	540	1080	
STCFD*	720	720	720
STCDF*	540	720	540
LDCIF	1260	1440	1440
LDCID	1260	1440	1440
LDCLF	1260	1980	
LDCLD	1260	1980	
LDEXP	540	900	
STCFI*	1200	1620	
STCFL*	1260	2160	
STCDI*	1260	1620	
STCDL*	1260	2160	
STEXP*	360	360	
	M0	**Not M0**	
CLRF	180	2150	
CLRD	180	4350	
NEGF	360	2400	
NEGD	360	2400	
ABSF	360	2400	
ABSD	360	2400	
TSTF	180	180	
TSTD	180	180	
LDFPS	180	0	
STFPS*	0		
STST*	0		
CFCC	0		
SETF	180		
SETD	180		
SETI	180		
SETL	180		

* Store Class Instructions

Load Class Instructions are those which do not deposit results in a memory location.

Execution of a Load Class Floating Point instruction by the Floating Point occurs in parallel with CPU operation and hence can be overlapped. Figure C-1 gives a simplified picture of how a Load Class Floating Point instruction is executed.

Store Class Instructions are those which store a result from the Floating Point into a memory location. Execution of a Store Class Instruction by the Floating Point Processor must occur before the result can be stored, hence parallel processing cannot occur for Store Class Floating Point Instructions.

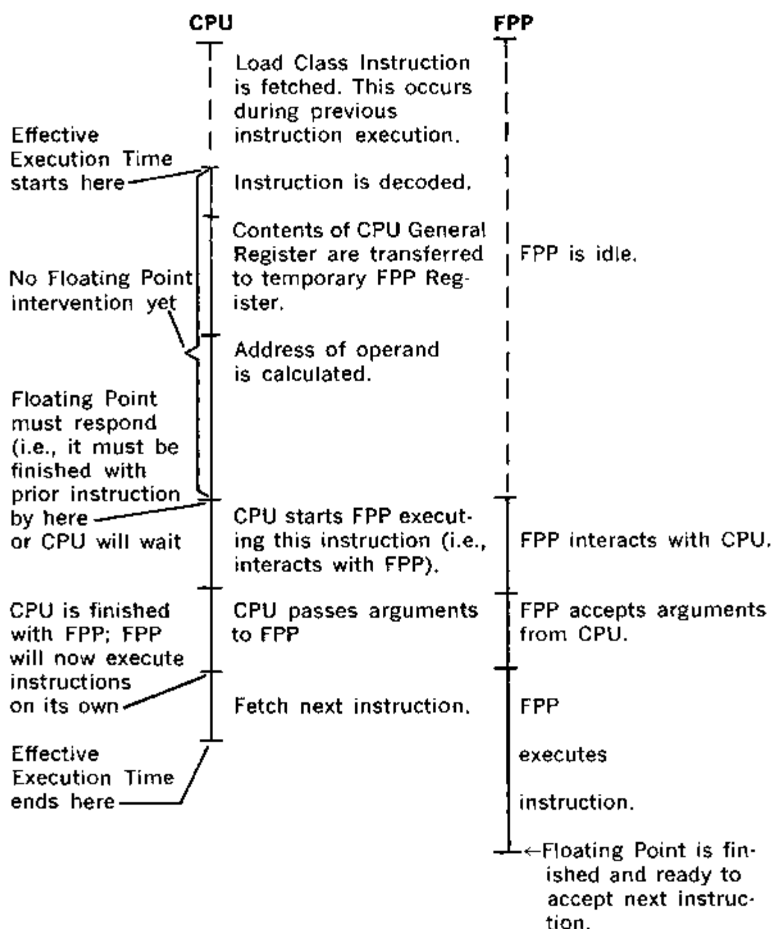


Figure C-1 Load Class Floating Point Instruction.

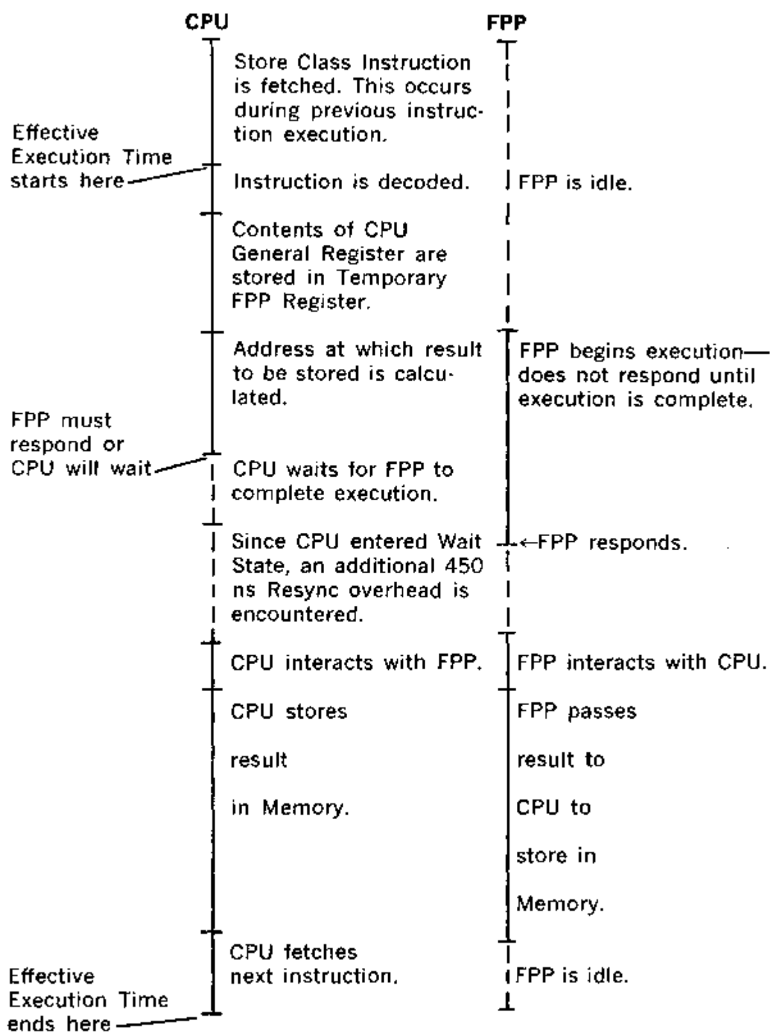


Figure C-2 Store Class Floating Point Instruction.

Figures C-1 and C-2 show, respectively, how timing associated with a typical Load Class and Store Class instruction is derived.

Figures C-3 and C-4 show, pictorially, how Effective Execution Times for actual Floating Point instructions in a program are calculated. Note that Effective Execution Times are dependent on previous Floating Point instruction.

Referencing Figure C-3, a sample calculation of Effective time would be:
for MULF (R0), AC1

Effective Execution Time is the summation of the following:

Preinteraction Time	450 ns
Address Calculation Time (Mode 1 from Table 11.1)	300 ns
Wait Time (Since FPP is idle, Wait = 0)	0 ns
Resync Time (Since Wait = 0, Resync = 0)	0 ns
Interaction Time	300 ns
Argument Transfer Time (Transfer 2 words @ 300 ns/word)	600 ns
Disengage and Fetch Time	300 ns
Effective Execution Time	1950 ns

for LDF X(R3),AC0 (Ref. Figure C-3)

First we calculate Wait Time:

Wait Time = [Floating Point Execution (previous FP instruction) (MULF)]	1800 ns
— [Disengage and Fetch Time (previous FPT instruction)]	– 300 ns
— [Execution Time of interposing nonFPT instruction (SOB)]	– 750 ns
— [Preinteraction Time]	– 450 ns
— [Address Calculation (Mode 6 from Table C-2)]	– 600 ns
	– 300 ns

Since calculation resulted in a negative number, Wait Time = 0.

. . . so Effective Execution Time is the summation of the following:

Preinteraction Time	450 ns
Address Calculation Time (Mode 6 from Table 11.1)	600 ns
Wait Time (From above calculation)	0 ns
Resync Time (Since Wait Time = 0, Resync = 0)	0 ns
Interaction Time	300 ns
Argument Transfer Time (2 words @ 300 ns/word)	600 ns
Disengage and Fetch Time	300 ns
Effective Execution Time	2250 ns

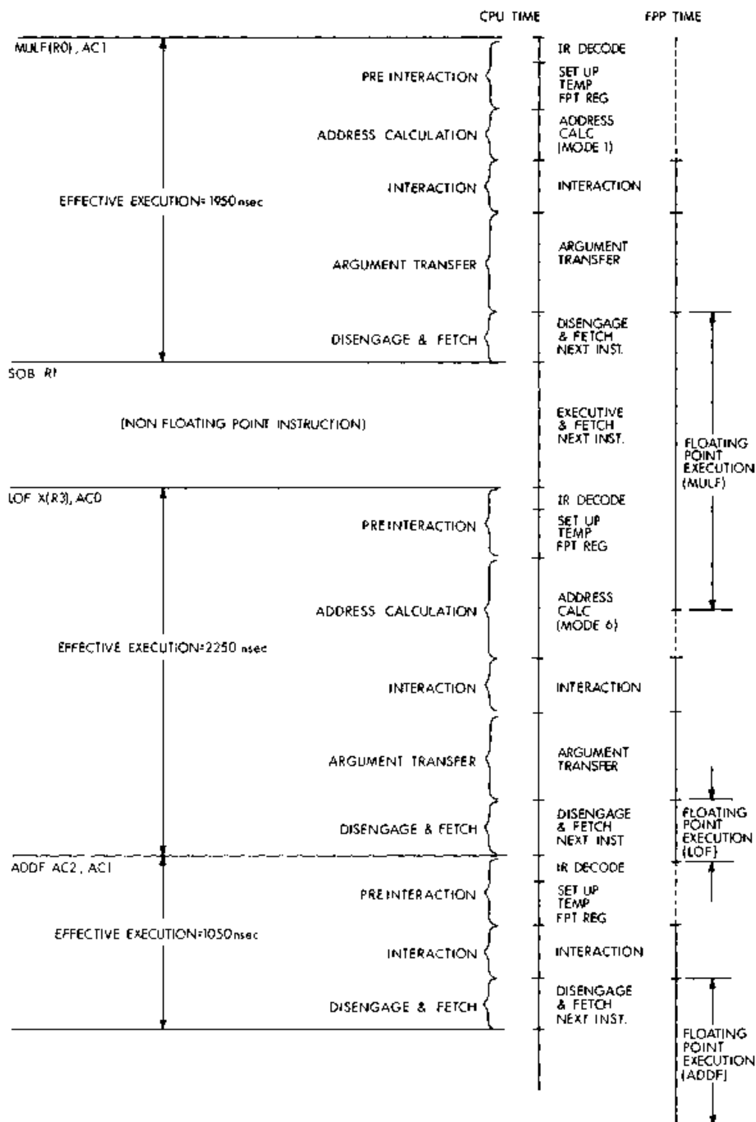


Figure C-3 Calculation of Effective Execution Times for Load Class Instructions

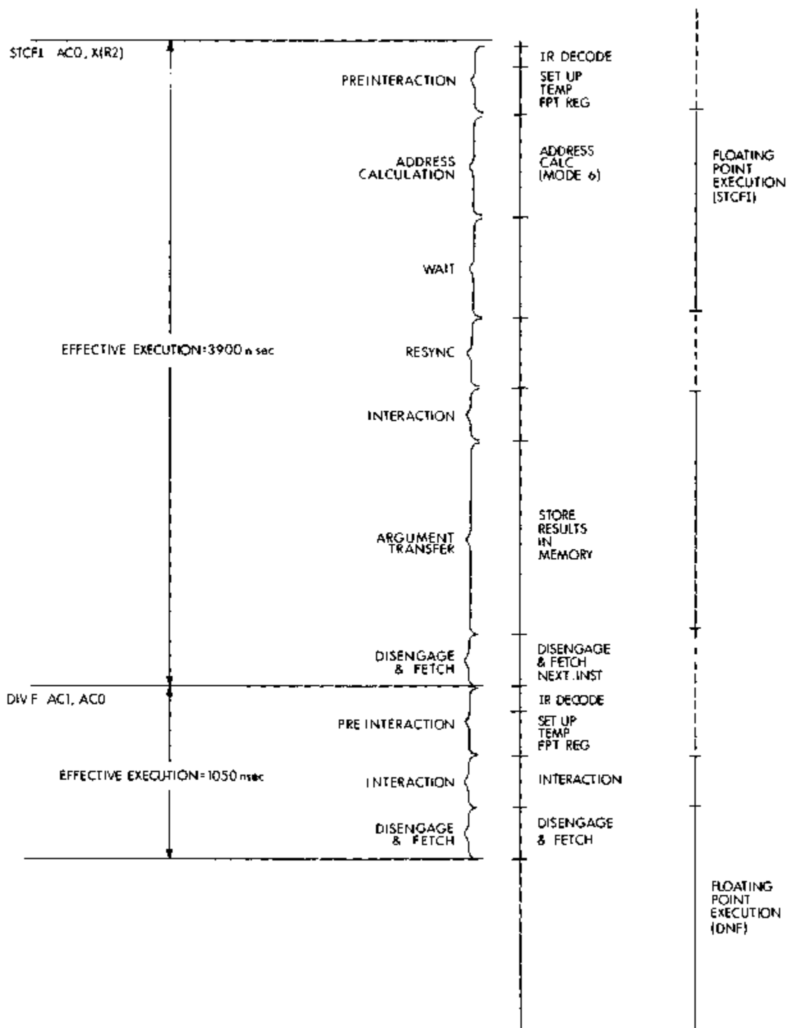


Figure C-4 Calculation of Effective Execution Time for Store Class Instructions

digital

DIGITAL EQUIPMENT CORPORATION, Corporate Headquarters: Maynard, Massachusetts 01754, Telephone: (617)897-5111 — SALES AND SERVICE OFFICES: UNITED STATES — ALABAMA, Huntsville • ARIZONA, Phoenix and Tucson • CALIFORNIA, El Segundo, Los Angeles, Oakland, Ridgecrest, San Diego, San Francisco (Mountain View), Santa Ana, Santa Clara, Stanford, Sunnyvale and Woodland Hills • COLORADO, Englewood • CONNECTICUT, Fairfield and Meriden • DISTRICT OF COLUMBIA, Washington (Lanham, MD) • FLORIDA, Ft. Lauderdale and Orlando • GEORGIA, Atlanta • HAWAII, Honolulu • ILLINOIS, Chicago (Rolling Meadows) • INDIANA, Indianapolis • IOWA, Bettendorf • KENTUCKY, Louisville • LOUISIANA, New Orleans (Metairie) • MARYLAND, Odenton • MASSACHUSETTS, Marlborough, Waltham and Westfield • MICHIGAN, Detroit (Farmington Hills) • MINNESOTA, Minneapolis • MISSOURI, Kansas City (Independence) and St. Louis • NEW HAMPSHIRE, Manchester • NEW JERSEY, Cherry Hill, Fairfield, Metuchen and Princeton • NEW MEXICO, Albuquerque • NEW YORK, Albany, Buffalo (Cheektowaga), Long Island (Huntington Station), Manhattan, Rochester and Syracuse • NORTH CAROLINA, Durham/Chapel Hill • OHIO, Cleveland (Euclid), Columbus and Dayton • OKLAHOMA, Tulsa • OREGON, Eugene and Portland • PENNSYLVANIA, Allentown, Philadelphia (Bluebell) and Pittsburgh • SOUTH CAROLINA, Columbia • TENNESSEE, Knoxville and Nashville • TEXAS, Austin, Dallas and Houston • UTAH, Salt Lake City • VIRGINIA, Richmond • WASHINGTON, Bellevue • WISCONSIN, Milwaukee (Brookfield) • INTERNATIONAL — ARGENTINA, Buenos Aires • AUSTRALIA, Adelaide, Brisbane, Canberra, Melbourne, Perth and Sydney • AUSTRIA, Vienna • BELGIUM, Brussels • BOLIVIA, La Paz • BRAZIL, Rio de Janeiro and Sao Paulo • CANADA, Calgary, Edmonton, Halifax, London, Montreal, Ottawa, Toronto, Vancouver and Winnipeg • CHILE, Santiago • DENMARK, Copenhagen • FINLAND, Helsinki • FRANCE, Lyon, Grenoble and Paris • GERMAN FEDERAL REPUBLIC, Cologne, Frankfurt, Hamburg, Hannover, Munich, Nuremburg, Stuttgart and West Berlin • HONG KONG • INDIA, Bombay • INDONESIA, Djakarta • IRELAND, Dublin • ITALY, ITALY, Milan, Rome and Turin • IRAN, Tehran • JAPAN, Osaka and Tokyo • MALAYSIA, Kuala Lumpur • MEXICO, Mexico City • NETHERLANDS, Utrecht • NEW ZEALAND, Auckland and Christchurch • NORWAY, Oslo • PUERTO RICO, Santurce • SINGAPORE • SPAIN, Madrid • SWEDEN, Gothenburg and Stockholm • SWITZERLAND, Geneva and Zurich • UNITED KINGDOM, Birmingham, Bristol, Epsom, Edinburgh, Leeds, Leicester, London, Manchester and Reading • VENEZUELA, Caracas •

PRINTED IN USA EB 05962 20/79 05C 02/09 03