



digital pdp-11

digital equipment corporation maynard, massachusetts

ADDRESS REGISTER

NUM BUS PWD-500

DATA

SOURCE DESTINATION ADDRESS

KEYBOARD

OFF

POWER

POWER

LOCK

17

16

15

14

13

12

11

10

9

8

7

6

5

4

3

2

1

0

17

16

15

14

13

12

11

10

9

8

7

6

5

4

3

2

1

0

D BRUNNER

pdp11  
handbook

digital

digital equipment corporation

Copyright 1969 by  
Digital Equipment Corporation

PDP is a registered trademark  
of Digital Equipment Corporation

The material in this handbook is for information purposes only and is subject to change without notice.



# TABLE OF CONTENTS

|   |    |
|---|----|
| <b>CHAPTER 1 INTRODUCTION</b>             |    |
| PDP-11 SYSTEMS .....                      | 1  |
| UNIBUS .....                              | 1  |
| KA11 PROCESSOR .....                      | 1  |
| Priority Interrupts .....                 | 1  |
| Reentrant Code .....                      | 2  |
| General Registers .....                   | 2  |
| Instruction Set .....                     | 2  |
| Addressing .....                          | 2  |
| Asynchronous Operation .....              | 2  |
| PACKAGING .....                           | 2  |
| SOFTWARE .....                            | 3  |
| <b>CHAPTER 2 SYSTEM INTRODUCTION</b>      |    |
| SYSTEM DEFINITION .....                   | 5  |
| SYSTEM COMPONENTS .....                   | 5  |
| UNIBUS .....                              | 5  |
| Single Bus .....                          | 5  |
| Bidirectional Lines .....                 | 5  |
| Master-Slave Relation .....               | 5  |
| Interlocked Communication .....           | 5  |
| Dynamic Master-Slave Relation .....       | 6  |
| KA11 CENTRAL PROCESSOR .....              | 6  |
| General Registers .....                   | 6  |
| Central Processor Status Register .....   | 6  |
| CORE MEMORY .....                         | 6  |
| PERIPHERAL DEVICES .....                  | 7  |
| SYSTEM INTERACTION .....                  | 7  |
| TRANSFER OF BUS MASTER .....              | 7  |
| PRIORITY STRUCTURE .....                  | 7  |
| NPR Requests .....                        | 8  |
| Interrupt Requests .....                  | 8  |
| <b>CHAPTER 3 ADDRESSING MODES</b>         |    |
| INTRODUCTION .....                        | 11 |
| ADDRESS FIELDS .....                      | 11 |
| General Register Addressing .....         | 11 |
| Indexed Addressing .....                  | 12 |
| Autoincrement Mode Addressing .....       | 12 |
| Autodecrement Addressing .....            | 12 |
| STACK PROCESSING .....                    | 13 |
| USE OF THE PC AS A GENERAL REGISTER ..... | 13 |
| Immediate Addressing .....                | 13 |
| Absolute Addressing .....                 | 13 |
| Relative Addressing .....                 | 14 |
| Deferred Relative Addressing .....        | 14 |
| USE OF THE SP AS A GENERAL REGISTER ..... | 14 |
| DOUBLE OPERAND ADDRESSING .....           | 14 |
| <b>CHAPTER 4 INSTRUCTION SET</b>          |    |
| INSTRUCTION TIMING .....                  | 17 |
| NOTATION .....                            | 17 |

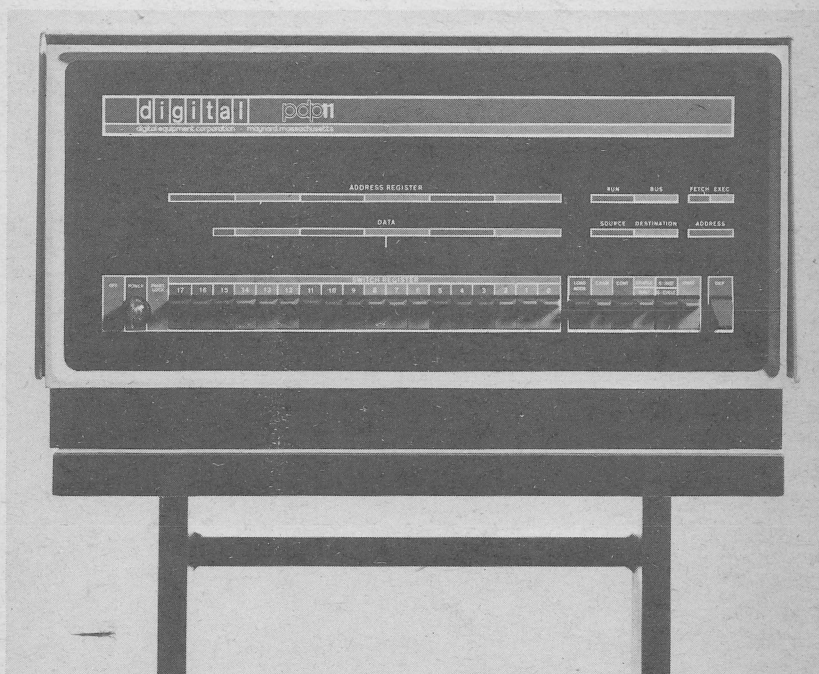
|   |    |
|---|----|
| <b>DOUBLE OPERAND INSTRUCTIONS</b> .....        | 17 |
| Arithmetic Operations .....                     | 18 |
| Boolean Instructions .....                      | 20 |
| <b>BRANCHES</b> .....                           | 21 |
| Unconditional Branch .....                      | 21 |
| Simple Conditional Branches .....               | 22 |
| Signed Conditional Branches .....               | 23 |
| Unsigned Conditional Branches .....             | 25 |
| JuMP .....                                      | 26 |
| <b>SUBROUTINES</b> .....                        | 27 |
| Examples .....                                  | 28 |
| <b>SINGLE OPERAND INSTRUCTIONS</b> .....        | 31 |
| Multiple Precision Operations .....             | 33 |
| Rotates .....                                   | 33 |
| Shifts .....                                    | 34 |
| Examples .....                                  | 36 |
| <b>BYTE OPERATIONS</b> .....                    | 36 |
| Double Operand Byte Instructions .....          | 36 |
| Example .....                                   | 37 |
| Single Operand Instructions .....               | 38 |
| <b>CONDITION CODE OPERATORS</b> .....           | 40 |
| <b>MISCELLANEOUS CONTROL INSTRUCTIONS</b> ..... | 41 |
| <b>PROCESSOR TRAPS</b> .....                    | 41 |
| Trap Instructions .....                         | 41 |
| Stack Overflow Trap .....                       | 43 |
| Bus Error Traps .....                           | 43 |
| Trace Traps .....                               | 43 |
| <br><b>CHAPTER 5 ADDRESS ALLOCATION</b>         |    |
| <b>ADDRESS MAP</b> .....                        | 45 |
| Interrupt and Trap Vector .....                 | 46 |
| Processor Stack and General Storage .....       | 46 |
| Peripheral Registers .....                      | 46 |
| <b>CORE MEMORY</b> .....                        | 46 |
| Read-Write Core Memory .....                    | 46 |
| Read-Only Core Memory .....                     | 46 |
| Wordlet Memory .....                            | 46 |
| <br><b>CHAPTER 6 PROGRAMMING OF PERIPHERALS</b> |    |
| <b>DEVICE REGISTERS</b> .....                   | 47 |
| <b>CONTROL &amp; STATUS REGISTERS</b> .....     | 47 |
| Device Function Bits .....                      | 48 |
| Memory Extension .....                          | 48 |
| Done Enable and Interrupt Enable .....          | 48 |
| Condition Bits .....                            | 48 |
| Unit Bits .....                                 | 48 |
| Error Bits .....                                | 48 |
| <b>DATA BUFFER REGISTERS</b> .....              | 49 |
| <b>PROGRAMMING EXAMPLES—NON INTERRUPT</b> ..... | 49 |
| <b>INTERRUPT STRUCTURE</b> .....                | 50 |
| <b>PROGRAMMING EXAMPLE</b> .....                | 51 |
| <br><b>CHAPTER 7 PERIPHERAL BULLETINS</b>       |    |
| <b>TELETYPE (MODEL LT33-DC/DD)</b> .....        | 53 |
| Size .....                                      | 53 |
| Power Requirement .....                         | 53 |

|   |    |
|---|----|
| <b>TELETYPE CONTROL (MODEL KL11)</b> .....                  | 53 |
| Teletype Control .....                                      | 53 |
| Keyboard/Reader Operation .....                             | 53 |
| Registers (TKS, TKB) .....                                  | 54 |
| Teleprinter/Punch .....                                     | 54 |
| Registers (TPS, TPB) .....                                  | 55 |
| Programming Example .....                                   | 55 |
| Peripheral Address Assignments .....                        | 55 |
| Mounting .....  | 55 |
| <b>HIGH-SPEED PERFORATED TAPE READER (MODEL PC11)</b> ..... | 55 |
| Tape Reader .....   | 55 |
| Registers (PRS, PRB) .....                                  | 56 |
| Programming Example .....                                   | 56 |
| Peripheral Address Assignments .....                        | 56 |
| Tape Punch .....  | 56 |
| Registers (PPS, PPB) .....                                  | 57 |
| Programming Example .....                                   | 57 |
| Peripheral Address Assignments .....                        | 57 |
| Mounting .....  | 57 |
| Environmental .....   | 58 |
| Line Frequency Clock (Model KW11-L) .....                   | 58 |
| Register .....  | 58 |
| Peripheral Address Assignments .....                        | 58 |
| Mounting .....  | 58 |
| Vector Address .....  | 58 |
| Priority Level .....  | 58 |
| <br><b>CHAPTER 8 DESCRIPTION OF THE UNIBUS</b>              |    |
| <b>GENERAL CONCEPTS OF THE UNIBUS</b> .....                 | 59 |
| Single Bus .....  | 59 |
| Bidirectional Bus .....                                     | 59 |
| Master-Slave Relation .....                                 | 59 |
| Interlocked Communication .....                             | 60 |
| Dynamic Master-Slave Relation .....                         | 60 |
| <b>UNIBUS SIGNALS</b> .....                                 | 60 |
| <b>NON-INTERRUPT SIGNALS</b> .....                          | 60 |
| Data Lines .....  | 60 |
| Address Lines .....   | 60 |
| Control Lines .....   | 61 |
| Master Sync & Slave Sync .....                              | 61 |
| Parity Available & Parity Bit .....                         | 61 |
| Initialization .....  | 61 |
| Spare 1 & Spare 2 .....                                     | 61 |
| <b>INTERRUPT SIGNALS</b> .....                              | 61 |
| Bus Request Lines .....                                     | 61 |
| Bus Grant Lines .....                                       | 61 |
| Non-Processor Request .....                                 | 61 |
| Non-Processor Grant .....                                   | 61 |
| Selection Acknowledge .....                                 | 61 |
| Interrupt (and) Bus Busy .....                              | 61 |
| <b>UNIBUS DATA TRANSFER OPERATIONS</b> .....                | 61 |
| DATO and DATOB .....  | 62 |
| DATI and DATIP .....  | 62 |
| Examples of Data Transfers .....                            | 62 |
| Signal Description of Data Transfers .....                  | 63 |



|   |    |
|---|----|
| UNIBUS CONTROL .....                                      | 64 |
| Priority Arbitration .....                                | 64 |
| Selection of Next Bus Master .....                        | 65 |
| Interrupt Sequence .....                                  | 65 |
| Example of Interrupt, etc. ....                           | 66 |
| Example of NPR Operation .....                            | 66 |
| <b>CHAPTER 9 INTERFACING</b>                              |    |
| REGISTERS .....   | 69 |
| BUS DRIVERS AND RECEIVERS .....                           | 69 |
| ADDRESS SELECTOR .....                                    | 71 |
| INTERRUPT CONTROL .....                                   | 74 |
| DEVICE CONTROL LOGIC .....                                | 76 |
| <b>CHAPTER 10 CONFIGURATION AND INSTALLATION PLANNING</b> |    |
| MODULAR CONSTRUCTION .....                                | 77 |
| MOUNTING BOXES AND CABINETS .....                         | 77 |
| PDP-11 Tabletop Box for 11/20, Etc. ....                  | 77 |
| PDP-11 Basic Mounting Box .....                           | 78 |
| PDP-11 Tabletop Extension Mounting Box .....              | 80 |
| PDP-11 Freestanding Base Cabinet .....                    | 80 |
| Freestanding Programmer's Table .....                     | 81 |
| SYSTEM UNITS AND CABLES .....                             | 81 |
| Peripheral Mounting Unit .....                            | 81 |
| Blank System Unit .....                                   | 81 |
| Unibus Module .....                                       | 82 |
| Unibus Cable .....  | 82 |
| CABLE REQUIREMENTS .....                                  | 82 |
| PDP-11/20 POWER REQUIREMENTS .....                        | 82 |
| TELETYPE REQUIREMENTS .....                               | 82 |
| ENVIRONMENTAL REQUIREMENTS .....                          | 82 |
| INSTALLATION PROCEDURE .....                              | 83 |
| <b>CHAPTER 11 PAPER TAPE SOFTWARE SYSTEM</b>              |    |
| PTS FEATURES .....  | 85 |
| PAL-11A Assembler .....                                   | 85 |
| ED11 Editor .....   | 85 |
| ODT On-Line Debugging .....                               | 85 |
| IOX Input/Output, etc. ....                               | 86 |
| Math Package .....  | 86 |
| Loaders .....   | 86 |
| Core Dump Routines .....                                  | 86 |
| <b>CHAPTER 12 THE OPERATOR'S CONSOLE</b>                  |    |
| CONSOLE ELEMENTS .....                                    | 87 |
| Indicator Lights .....                                    | 87 |
| Register Displays .....                                   | 87 |
| Switch Register .....                                     | 87 |
| Control Switches .....                                    | 87 |
| CONTROL SWITCH OPERATION .....                            | 89 |
| <b>APPENDIX A—PDP-11 INSTRUCTION REPERTOIRE</b> .....     | 91 |
| <b>APPENDIX B—ADDRESSING SUMMARY</b> .....                | 95 |
| ADDRESSING MODES .....                                    | 95 |
| General Register Addressing .....                         | 95 |
| PC Register Addressing .....                              | 95 |

|   |     |
|---|-----|
| INSTRUCTION FORMATS .....                 | 95  |
| APPENDIX C—ADDRESS MAP .....              | 97  |
| APPENDIX D—UNIBUS OPERATIONS .....        | 99  |
| DATA-TRANSFERS .....                      | 99  |
| DATI and DATIP .....                      | 99  |
| DATO and DATOB .....                      | 100 |
| PTR-PRIORITY TRANSFER .....               | 101 |
| INTR—INTerRupt .....                      | 102 |
| GENERAL NOTES ON THE BUS OPERATIONS ..... | 102 |



The PDP-11 is available in two versions—PDP-11/10 and PDP-11/20. The basic PDP-11/10 contains 1,024 words of read only memory in conjunction with 128 words of read/write memory and the basic PDP-11/20 includes 4,096 words of read/write memory.



# CHAPTER 1

## INTRODUCTION

This publication is a handbook for Digital Equipment Corporation's PDP-11. It provides a comprehensive overview of the system structure, the instruction repertoire, input/output programming, peripherals, general interfacing, software, and console operation.

PDP-11 is Digital's answer to the demand for a modular system for real-time data acquisition, analysis and control. PDP-11 systems can handle a wide variety of real-time control applications—each system being individually tailored from a comprehensive array of modular building blocks. Digital is unique among manufacturers of small-scale computers in its ability to provide not only fast and efficient processing units, but also a large family of its own compatible I/O devices including A/D and D/A converters, magnetic tape, disk storage, paper tape, and displays, as well as a wide range of general-purpose modules. This capability offers the user a new, more efficient approach to real-time systems.

The following paragraphs introduce the new PDP-11 by way of highlighting several of the important design features that set it apart from other machines in its class. Subsequent chapters of this manual place these features in their proper context and provide detailed descriptions of each.

### PDP-11 SYSTEMS

The PDP-11 is available in two versions designated as PDP-11/10 and PDP-11/20. The PDP-11/10 contains a KA11 processor, 1,024 words of 16-bit read-only memory, and 128 16-bit words of read-write memory. The basic PDP-11/20 contains a KA11 processor and 4,096 words of 16-bit read-write core memory, a programmer's console, and an ASR-33 Teletype. Both versions can be similarly expanded with either read-write or read-only memory and peripheral devices.

### UNIBUS

Unibus is the name given to the single bus structure of the PDP-11. The processor, memory and all peripheral devices share the same high-speed bus. The Unibus enables the processor to view peripheral devices as active memory locations which perform special functions. Peripherals can thus be addressed as memory. In other words, memory reference instructions can operate directly on control, status, or data registers in peripheral devices. Data transfers from input to output devices can bypass the processor completely.

### KA11 PROCESSOR

The KA11 processor incorporates a unique combination of powerful features not previously available in 16-bit computers.

**Priority Interrupts**—A four-level automatic priority interrupt system permits the processor to respond automatically to conditions outside the system, or in the processor itself. Any number of separate devices can be attached to each level.

Each peripheral device in a PDP-11 system has a hardware pointer to its own unique pair of memory locations which, in turn, point to the device's service routine. This unique identification eliminates the need for polling of devices

to identify an interrupt, since the interrupt servicing hardware selects and begins executing the appropriate service routine.

The device's interrupt priority and service routine priority are independent. This allows dynamic adjustment of system behavior in response to real-time conditions.

The interrupt system allows the processor continually to compare its own priority levels with the levels of any interrupting devices and to acknowledge the device with the highest level above the processor's priority level. Servicing an interrupt for a device can be interrupted for servicing a higher priority device. Service to the lower priority device can be resumed automatically upon completion of the higher level servicing. Such a process, called nested interrupt servicing, can be carried out to any level.

**Reentrant Code**—Both the interrupt handling hardware and the subroutine call hardware are designed to facilitate writing reentrant code for the PDP-11. This type of code allows use of a single copy of a given subroutine or program to be shared by more than one process or task. This reduces the amount of core needed for multi-task applications such as the concurrent servicing of many peripheral devices.

**General Registers**—The PDP-11 is equipped with eight general registers. All are program-accessible and can be used as accumulators, as pointers to memory locations, or as full-word index registers. Six registers are used for general-purpose access while the seventh and eighth registers are used as a stack pointer and program counter respectively.

**Instruction Set**—An important feature of the PDP-11 instruction set is the availability of double operand instructions. These instructions allow memory-to-memory processing and eliminate the need to use registers for storage of intermediate results. By using double operand instructions, every memory location can be treated as an accumulator. This significantly reduces the length of programs by eliminating load and store operations associated with single operand machines.

**Addressing**—Much of the power of the PDP-11 is derived from its wide range of addressing capabilities. PDP-11 addressing modes include list sequential addressing, full address indexing, full 16-bit word addressing, 8-bit byte addressing, stack addressing, and direct addressing to 32K words. Variable length instruction formatting allows a minimum number of bits to be used for each addressing mode. This results in efficient use of program storage space.

**Asynchronous Operation**—The PDP-11's memory and processor operations are asynchronous. As a result, I/O devices transferring directly to or from memory may steal memory cycles during instruction operation.

## PACKAGING

The PDP-11 has adopted a modular approach to allow custom configuring of systems, easy expansion, and easy servicing. Systems are composed of basic building blocks, called System Units, which are completely independent subsystems connected only by pluggable Unibus and power connections. There is no fixed wiring between them. An example of this type of subsystem is a 4,096-word memory module.

System Units can be mounted in many combinations within the PDP-11 hardware, since there are no fixed positions for memory or I/O device controllers. Additional units can be mounted easily and connected to the system

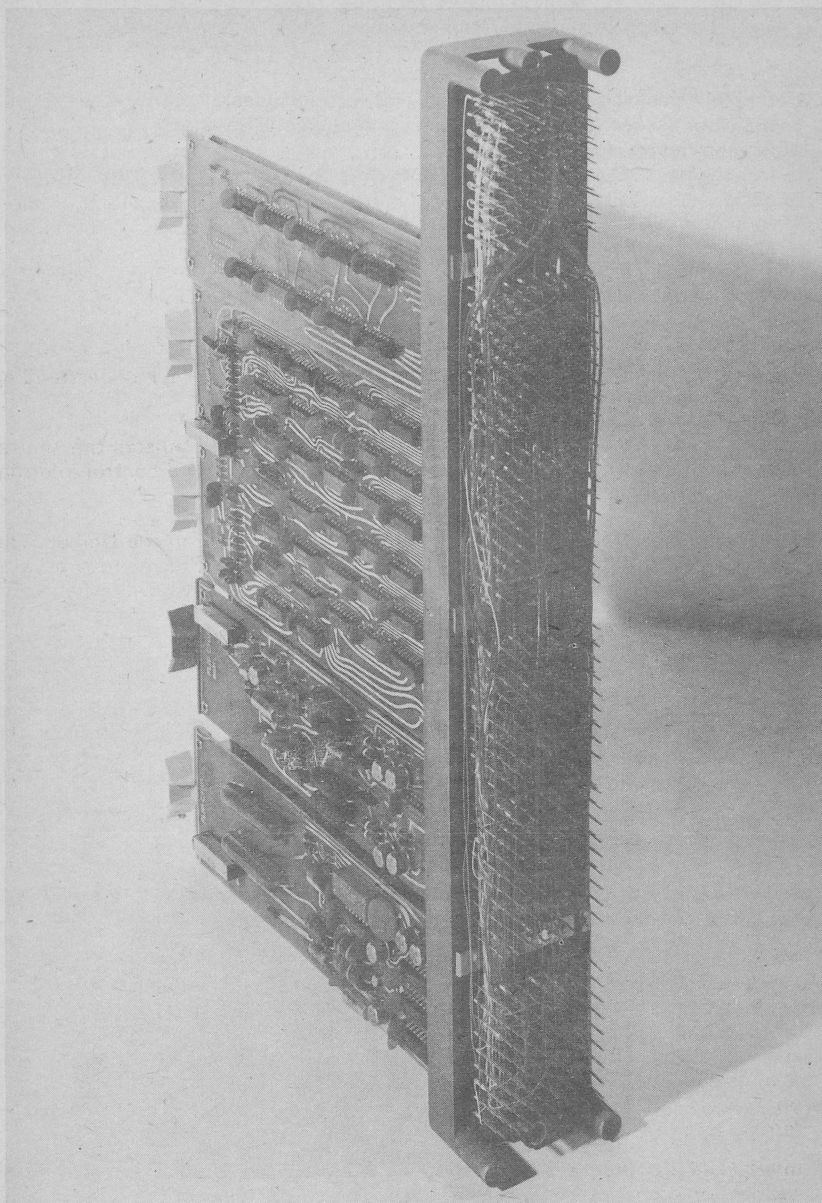
in the field. In case maintenance is required, defective System Units can be replaced with spares and operation resumed within a few minutes.

## **SOFTWARE**

A complete package of user-oriented software includes:

- Absolute assembler providing object and source listings
- String-oriented editor
- Debugging routines capable of operating in a priority interrupt environment
- Input/output handlers for standard peripherals
- Relocatable integer and floating point math library





All PDP-11 processors, memories and peripherals are electrically and mechanically modular subsystems supported in System Units which are simply plugged together to form a computer tailored to user needs.

# CHAPTER 2

## SYSTEM INTRODUCTION

### SYSTEM DEFINITION

Digital Equipment Corporation's PDP-11 is a 16-bit, general-purpose, parallel-logic computer using two's complement arithmetic. The PDP-11 is a variable word length processor which directly addresses 32,768 16-bit words or 65,536 8-bit bytes. All communication between system components is done on a single high-speed bus called a Unibus. Standard features of the system include eight general-purpose registers which can be used as accumulators, index registers, or address pointers, and a multi-level automatic priority interrupt system.

### SYSTEM COMPONENTS

**UNIBUS**—There are five concepts that are very important for understanding both the hardware and software implications of the Unibus.

**Single Bus**—The Unibus is a single, common path that connects the central processor memory, and all peripherals. Addresses, data, and control information are sent along the 56 lines of the bus.

The form of communication is the same for every device on the Unibus. The processor uses the same set of signals to communicate with memory as with peripheral devices. Peripheral devices also use this set of signals when communicating with the processor, memory, or other peripheral devices.

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.

**Bidirectional Lines**—Unibus lines are bidirectional, so that the same signals which are received as input can be driven as output. This means that a peripheral device register can be either read or set by the central processor or other peripheral devices; thus, the same register can be used for both input and output functions.

**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.

**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 and the response time of the master and slave devices. The maximum transfer rate on the Unibus is one 16-bit word every 750 nanoseconds, or 1.3 million 16-bit words per second.



Read-only core memory (ROM) is available in 1,024 16 bit-word segments. The access time of the ROM is 500 nanoseconds. Memory is also available in 128 16-bit word segments with a 2.0 microsecond cycle time. Both 1,024 words of read-only memory and 128 words of read-write memory mount in a single System Unit and are a standard part of the PDP-11/10 system.

**PERIPHERAL DEVICES**—The ASR-33 Teletype with low-speed paper tape reader and punch is provided in the basic PDP-11/20 system. Options for the PDP-11 include a paper tape reader capable of reading 300 characters per second, a paper tape punch with an output capacity of 50 characters per second, and additional Teletype units. Provision is made for the addition of numerous peripheral devices. These include standard DEC peripherals as well as other devices which will be unique to the PDP-11.

## SYSTEM INTERACTION

At any point in time only one device can be in control of the bus, or be bus master. The master communicates with another device on the bus which is called the slave. Usually, the established master will communicate with the slave in the form of data transfers.

Full 16-bit words or 8-bit bytes of information can be transferred on the bus between the master and the 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 restoring the results into memory after execution of instructions. Pure data transfers occur between a disk control and memory.

**TRANSFER OF BUS MASTER**—When a device (other than the central processor) is capable of becoming bus master and requests use of the bus, it is generally for one of two purposes: 1) to make a non-processor transfer of data directly to or from memory, or 2) to interrupt program execution and force the processor to branch to a specific address where an interrupt service routine is located.

**PRIORITY STRUCTURE**—When a device capable of becoming bus master requests use of the bus, the handling of that request depends on the location of that device in the priority structure. These factors must be considered to determine the priority of the request:

1. The processor's priority can be set under program control to one of eight levels using bits 7, 6, and 5 in the processor status register. These three bits set a priority level that inhibits granting of bus requests on lower levels.
2. Bus requests from external devices can be made on one of five request lines. A non-processor request (NPR) has the highest priority, and its request is honored by the processor between bus cycles of an instruction execution. Bus request 7 (BR7) is the next highest priority, and BR4 is the lowest. The four lower level priority requests are honored by the processor between instructions. When the processor's priority is set to a level, for example 6, all bus requests on BR6 and below are ignored.
3. When more than one device is connected to the same bus request (BR) line, a device nearer the central processor has a higher priority than a device farther away. Any number of devices can be connected to a given BR or NPR line.

Once a device other than the processor has control of the bus, it is for one of two types of requests: 1) NPR Request, 2) Interrupt Request.



**NPR Requests**—NPR data transfers can be made between any two peripheral devices without the supervision of the processor. Normally, NPR transfers are between a mass storage device, such as a disk, and core memory. The structure of the bus also permits device-to-device transfers, allowing customer-designed peripheral controllers to access other devices such as disks directly.

An NPR device has very fast access to the bus and can transfer at high data rates once it has control. The processor state is not affected by the transfer; therefore the processor can relinquish control while an instruction is in progress. This can occur at the end of any bus cycle except in between a read-modify-write sequence. (See Chapter 8 for details). In the PDP-11, an NPR device can gain bus control in 3.5 microseconds or less. An NPR device in control of the bus may transfer 16-bit words from memory at memory speed or every 1.2 microseconds in the PDP-11/20 or every 1.0 microseconds in the PDP-11/10.

**Interrupt Requests**—Devices that request interrupts on the bus request lines (BR7, BR6, BR5, BR4) can take advantage of the power and flexibility of the processor. The entire instruction set is available for manipulating data and status registers. When a device servicing program must be run, the task currently under way in the central processor is interrupted and the device service routine is initiated. Once the device request has been satisfied, the processor returns to the interrupted task.

In the PDP-11, the return address for the interrupted routine and the processor status word are held in a "stack." A stack is a dynamic sequential list of data with special provision for access from one end. A stack is also called a "push down" or "LIFO" (Last-In First-Out) list. Storage and retrieval from stacks is called "pushing" and "popping" respectively. These operations are illustrated in Figure 2-1.

In the PDP-11, a stack is automatically maintained by the hardware for interrupt processing. Thus, higher level requests can interrupt the processing of lower level interrupt service, and automatically return control to the lower level interrupt service routines when the higher level servicing is completed.

Here is an example of this procedure. A peripheral requires service and requests use of the bus at one of the BR levels (BR7, BR6, BR5, BR4). The operations undertaken to "service" the device are as follows:

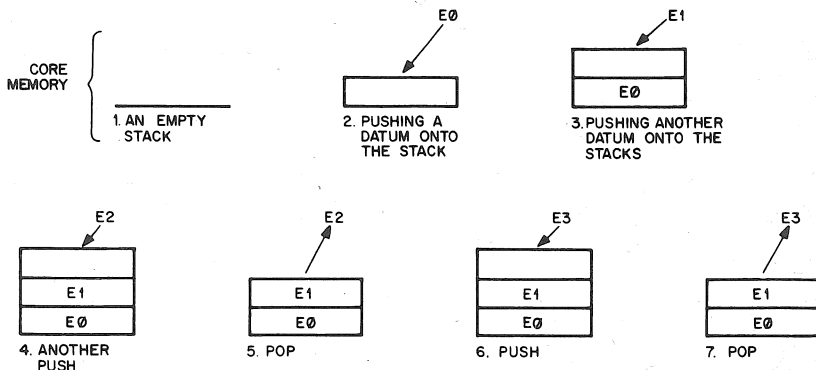


Fig 2-1 Illustration of Push and Pop Operations

1. Priorities permitting, the processor relinquishes the bus to that device.
2. When the device has control of the bus, it sends the processor an interrupt command with the address of the words in memory containing the address and status of the appropriate device service routine.
3. The processor then "pushes"—first, the current central processor status (PS) and then, the current program counter (PC) onto the processor stack.
4. The new PC and PS (the "interrupt vector") are taken from the location specified by the device and the next location, and the device

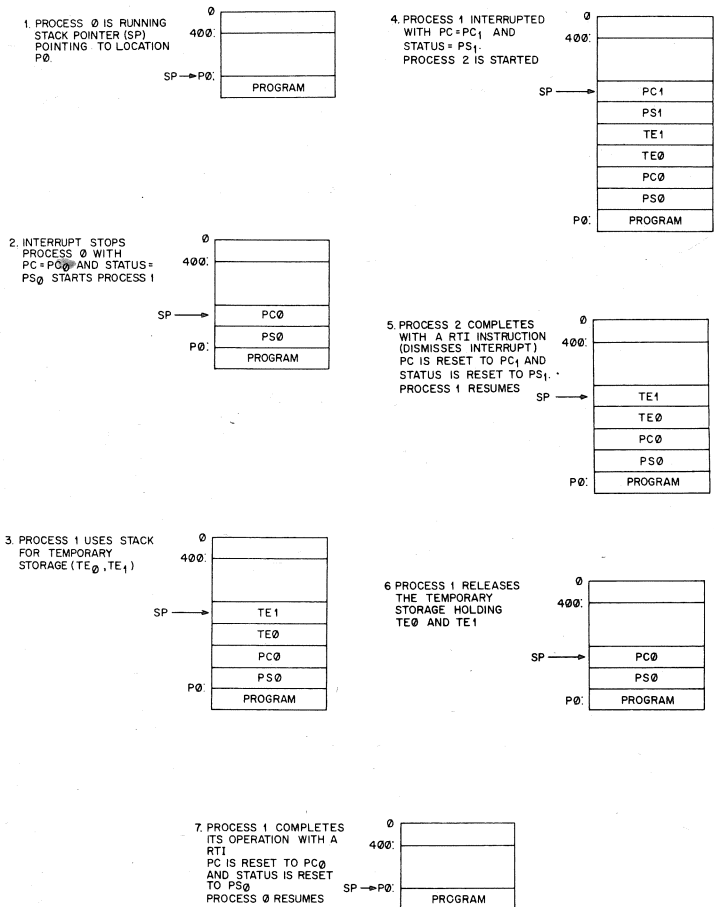


Figure 2-2 Nested Device Servicing

service routine is begun. Note that those operations all occur automatically and that no device-polling is required to determine which service routine to execute.

5. 7.2 microseconds is the time interval between the central processor's receiving the interrupt command and the fetching of the first instruction. This assumes there were no NPR transfers during this time.
6. The device service routine can resume the interrupted process by executing the RTI (Return from Interrupt) instruction which "pops" the processor stack back into the PC and PS. This requires 4.5 microseconds if there are no intervening NPR's.
7. A device service routine can be interrupted in turn by a sufficiently high priority bus request any time after completion of its first instruction.
8. If such an interrupt occurs, the PC and PS of the device service routine are automatically pushed into the stack and the new device routine initiated as above. This "nesting" of priority interrupts can go on to any level, limited only by the core available for the stack. More commonly, this process will nest only four levels deep since there are four levels of BR signals. An example of nested device servicing is shown in Figure 2-2. A rough core map is given for each step of the process. The SP points to the top word of the stack as shown.

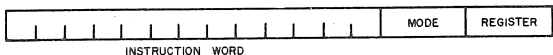
## CHAPTER 3

Most data in a program is structured in some way—in a table, in a stack, in a table of addresses, or perhaps in a small set of frequently-used variables local to a limited region of a program. The PDP-11 handles these common data structures with addressing modes specifically designed for each kind of access. In addition, addressing for unstructured data is general enough to permit direct random access to all of core. Memory is not broken up into pages and fields (often awkward and wasteful of core storage).

Addressing in the PDP-11 is done through the general registers. Programs requiring several stacks can use the general registers for stack pointers. Those requiring many local variables can use general registers as accumulators. The general registers can be used interchangeably as index registers or as sequential list pointers to access tabular data. Address arithmetic may be done directly in the general registers.

## ADDRESS FIELDS

PDP-11 instruction words contain a 6-bit address field divided into two sub-fields selecting the general register and the mode generating the effective address.



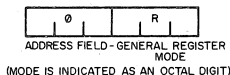
The register subfield specifies which of the eight general registers is to be used in the address calculation. The mode subfield indicates how this register is to be used in determining the operand. These modes will be described in the following paragraphs.

**GENERAL REGISTER ADDRESSING**—The general registers will be used as simple accumulators for operating on frequently-accessed variables. In this mode, the operand is held directly in the general register. The general registers are in fast memory, (280-nanosecond cycle time) resulting in a speed improvement for operations on these variables.

PAL-11, the PDP-11 assembler, interprets instructions of the form

OPR R

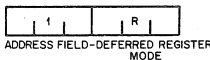
as general register operations. R has been defined as a register name and OPR is used to represent a general instruction mnemonic. The address field for general register operations is



Operands that are pointed to by addresses (indirect or deferred) are denoted to PAL-11 by the @ symbol. Thus, instructions of the form

OPR @R

specify deferred register addressing and have the following address field.



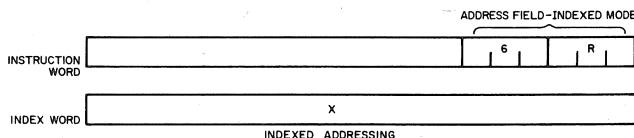
Deferred register addressing may also be selected in PAL-11 by the form OPR (R).

**INDEXED ADDRESSING**—The general registers may be used as index registers to permit random access of items in tables or stacks of data. Instructions of the form

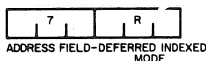
OPR X(R)

specify indexed mode addressing. The effective address is the sum of X and the contents of the specified general register R.

The index word containing X follows the instruction word.



Index mode addressing can be deferred to permit access of data elements through tables or stacks of their addresses. The address field for index deferred mode is



It is specified by instructions of the form

OPR @X(R)

**AUTOINCREMENT ADDRESSING**—Autoincrement addressing provides for automatic stepping of a pointer through sequential elements of a table of operands. In this mode, the address of the operand is taken from the general register and then the contents of the register are stepped (incremented by one or two) to address the next word or byte depending upon whether the instruction operates on byte or word data. Instructions of the form

OPR (R)+

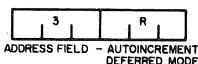
specify autoincrement addressing. The address field for autoincrement addressing is



This mode may also be deferred. Instructions of the form

OPR @(R)+

specify deferred autoincrement addressing and assemble with the following address field.

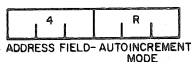


**AUTODECREMENT ADDRESSING**—Autodecrement addressing steps the specified general register to the next lower byte (decrement by one) or word

(decrement by two) address and uses the new contents of the general register as the operand address. Instructions of the form

OPR —(R)

specify autodecrement addressing. The address field for autodecrement addressing is



This mode also may be deferred and specified by instructions of the form OPR @ —(R). When deferred the address field is



## STACK PROCESSING

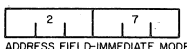
The combination of autoincrement addressing in which the general register is stepped forward *after* the operand address is determined and autodecrement addressing in which the general register is stepped backward *before* the operand address is determined is the basic requirement for convenient low overhead stack operations.

The PDP-11 has extensive stack processing capabilities. The stack pointer (SP), R6, maintains a stack for the nested handling of interrupts. All of the general registers can maintain stacks under program control. Elements in the middle of stacks may be accessed through indexed addressing. This provides for convenient access of dynamically assigned temporary storage, especially useful in nested procedures.

## USE OF THE PC AS A GENERAL REGISTER

There are special implications in the use of the addressing modes already described when applied to the PC (R7). The use of the PC with the addressing modes described above generates immediate, absolute, relative, and deferred relative addressing.

**IMMEDIATE ADDRESSING**—Immediate addressing provides time and space improvement for access of constant operands by including the constant in the instruction. The instruction word referencing an immediate operand specifies autoincrement addressing through the program counter. The address field would be



The program counter points to the word after the instruction word. The contents of this word are therefore used as the operand and the program counter is stepped to the next word. PAL-11 recognizes address expressions of the form "#n" as immediate operands and codes them with the address field shown above followed by a word of data (n).

A full word is assembled for immediate operands even in byte instructions so that instruction words are always fetched from even locations.

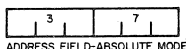
**ABSOLUTE ADDRESSING**—The contents of the location following the instruc-



tion word may be taken as the address of an operand by specifying deferral in immediate mode addressing. That is, instructions of the form

OPR @ #A

refer to the operand at address A. PAL-11 assembles address expressions of this form into an address field

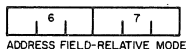


followed by a word containing the operand address.

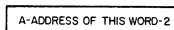
**RELATIVE ADDRESSING**—Relative addressing specifies the operand address relative to the instruction location. This is accomplished by using the PC as an index register. The PC is considered as a base address. The offset, the distance between the location of the operand and the PC, is held in the index word of the instruction. PAL-11 assembles instructions of the form

OPR A

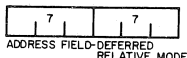
(where A has not been assigned as a name of a general register) as an instruction word with the address field



followed by an index word of the form



**DEFERRED RELATIVE ADDRESSING**—Deferral of relative addressing permits access to data through memory locations holding operand addresses. The "@" character specifies deferred addressing; i.e., OPR @A. The address field for deferred relative addressing is

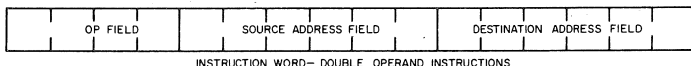


## USE OF THE SP AS A GENERAL REGISTER

The processor stack pointer will in most cases be the general register used in PDP-11 stack operations. Note that the content of SP, (SP), refers to the top element of the stack, that  $-(SP)$  will push data onto the stack, that  $(SP)+$  will pop data off the stack, and that  $X(SP)$  will permit 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 will simply leave odd addresses unmodified.

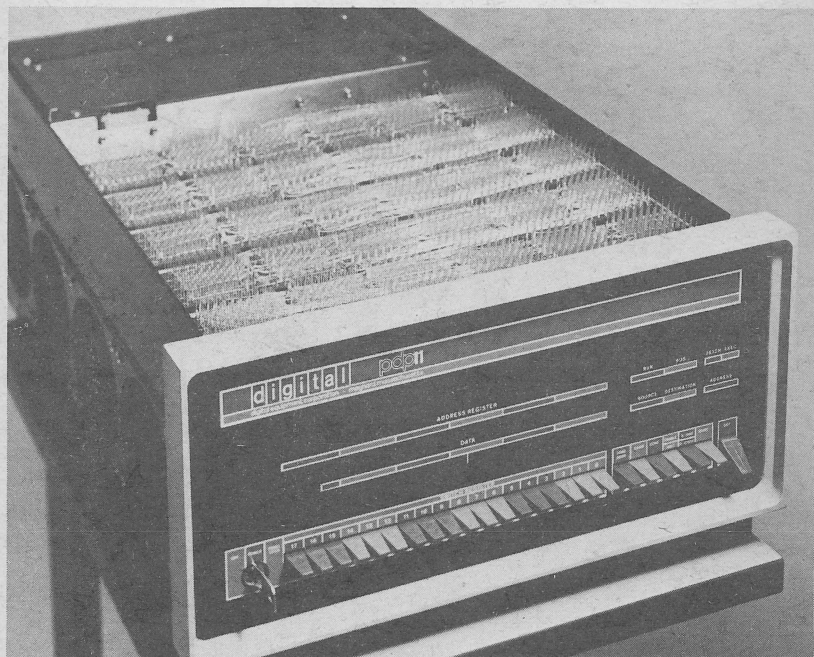
## DOUBLE OPERAND ADDRESSING

Operations which imply two operands such as add, subtract and compare are presented in the PDP-11 by instructions which specify two addresses. The instruction word for such operations is of the form



Instruction Word—Double Operand Instructions

and is followed by index words and immediate operands for the source and destination address fields as appropriate. Source address calculations are performed before destination address calculations. Since each operand may be anywhere in core storage or in the general registers, each memory location is thus effectively provided with the arithmetic capabilities of an accumulator. Further, since peripheral device registers and memory location are addressed in the same way, the contents of peripheral data buffers can be stored or loaded directly to and from memory without use of any general register. This means that interrupt routines can be executed without saving and restoring any of the general registers.



The PDP-11 is a 16-bit computer with a universal bus called a Unibus allowing networks of memories and peripherals to be used in virtually any combination.

## CHAPTER 4

# INSTRUCTION SET

This chapter presents the order code for the PDP-11. Each PDP-11 instruction is described in terms of five parameters: operation, effect on condition codes, base timing, assembler mnemonics, and octal representation. Special comments are included where appropriate.

### NOTATION

The following notations will be used in this section:

|               |                                       |
|---------------|---------------------------------------|
| (XXX)         | : The contents of XXX                 |
| src           | : The Source Address                  |
| dst           | : The Destination Address             |
| $\wedge$      | : Boolean "AND" Function              |
| $\vee$        | : Boolean "OR" Function               |
| $\nabla$      | : Boolean "Exclusive OR" Function     |
| $\sim$        | : Boolean "NOT" Function (Complement) |
| $\rightarrow$ | : "becomes"                           |
| $\uparrow$    | : "is popped from the stack"          |
| $\downarrow$  | : "is pushed onto the stack"          |

### INSTRUCTION TIMING

The PDP-11 is an asynchronous processor in which, in many cases, memory and processor operations are overlapped. The execution time for an instruction is the sum of a basic instruction time and the time to determine and fetch the source and/or destination operands. The following table shows the addressing times required for the various modes of addressing source and destination operands. The instruction time for each operation is given (throughout this chapter) for the 11/20 configuration. All times stated are subject to  $\pm 20\%$  variation.

#### ADDRESSING FORM

#### TIMING

| (src or dst)     | src ( $\mu$ s) <sup>†</sup> | dst ( $\mu$ s) <sup>†</sup> |
|------------------|-----------------------------|-----------------------------|
| R                | 0                           | 0                           |
| (R) or @R        | 1.5                         | 1.4*                        |
| (R) +            | 1.5                         | 1.4*                        |
| -(R)             | 1.5                         | 1.4*                        |
| @(R) +           | 2.7                         | 2.6*                        |
| @-(R)            | 2.7                         | 2.6*                        |
| BASE(R)          | 2.7                         | 2.6*                        |
| @BASE(R) or @(R) | 3.9                         | 3.8*                        |

\* dst time is .4  $\mu$ s. less than listed time if instruction was a  
CoMPare, CoMPare Byte  
Bit Test, Bit Test Byte  
TeST, or TeST Byte

none of which ever modify the destination word.

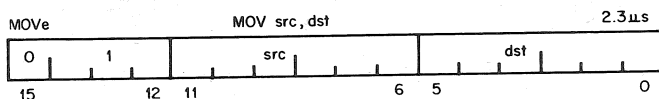
<sup>†</sup> referencing bytes at odd addresses adds 0.6 $\mu$ s to src and dst times.

**DOUBLE OPERAND INSTRUCTIONS**—Double Operand Instructions are represented in assembly language as:

OPR src, dst

where src and dst are the addresses of the source and destination operands respectively. The execution time for these operations is comprised of the source time, the destination time, and the instruction time. The source and destination times depend on addressing modes and are described in the preceding table.

## Arithmetic Operations—



Operation: (src) → (dst)

Condition Codes:

Z: set if (src) = 0; cleared otherwise

N: set if (src) < 0; cleared otherwise

C: not affected

V: cleared

Description: Moves the source operand to the destination location. The previous contents of the destination are lost. The contents of the source are not affected.

The MOV instruction is a generalization of 'load,' 'store,' 'setup,' 'push,' 'pop,' and interregister transfer operations.

General registers may be loaded with the contents of memory addresses with instructions of the form:

MOV src, R

Registers may be loaded with a counter, and pointer values with MOV instructions:

MOV #n, R

(which loads the number n into register R)

Operands may be pushed onto a stack by:

MOV src, -(R)

and may be popped off a stack by:

MOV (R)+, dst

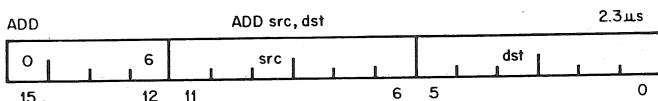
Interregister transfers are simply:

MOV RA, RB

(RA and RB are general registers)

Memory-to-memory transfers may be done with the MOV instruction in the general form:

MOV src, dst



Operation: (src) + (dst) → (dst)

Condition Codes:

Z: set if result = 0; cleared otherwise

N: set if result < 0; cleared otherwise

C: set if there was a carry from the most significant bit of the result; cleared otherwise

V: set if there was arithmetic overflow as a result of the operation, that is, if both operands were of the same sign and the result was of the opposite sign; 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.

The ADD instruction includes as special cases the "add-to-register," "add-to-memory," and "add-register-to-register" functions:

|                          |            |
|--------------------------|------------|
| Add-to-Register          | ADD src, R |
| Add-to-Memory            | ADD R, dst |
| Add Register-to-Register | ADD RA, RB |

Arithmetic may also be done directly in memory by the general form ADD instruction

ADD src, dst

Use of this form saves considerable loading and storing of accumulators.

Two special cases of the ADD instruction are particularly useful in compilers, interpreters, and other stack arithmetic processes:

ADD (R)+, (R)  
(where R is the stack pointer)

which replaces the top two elements of the stack with their sum; and ADD src, (R), which increases the top element of the stack by the contents of the source address.

The "Add Immediate" operation is yet another special case of this generalized ADD instruction:

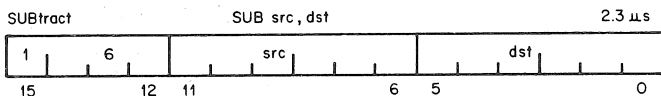
ADD #n, dst

Immediate operations are useful in dealing with constant operands. Note that:

ADD #n, R

steps the register R (which may be an index register) through n addresses eliminating the need for a special "add-to-index-register" instruction.

All these special cases of the ADD instruction apply equally well to the other double operand instructions that follow.



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

Condition Codes:

Z: set if result = 0; cleared otherwise

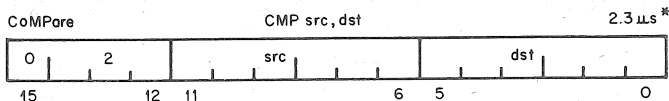
N: set if result < 0; cleared otherwise

C: cleared if there was a carry from the most significant bit of the result; set otherwise

V: set if there was arithmetic overflow as a result of the operation, that is, if the operands were of opposite signs and the sign of source was the same as the sign of the result; cleared 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.



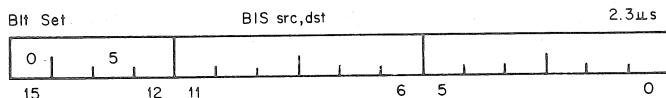


Operation: (src) — (dst) [in detail, (src) +  $\sim$  (dst) + 1]

Condition Codes:    Z: set if result = 0; cleared otherwise  
                           N: set if result < 0; cleared otherwise  
                           C: cleared if there was a carry from the most significant bit of the result; set 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.

Description: Arithmetically compares the source and destination operands. Affects neither operand. The only action is to set the condition codes appropriately.

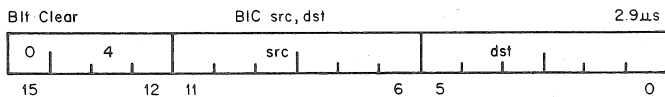
**Boolean Instructions**—These instructions have the same format as the double operand arithmetic group. They permit operations on data at the bit level.



Operation: (src) V (dst)  $\rightarrow$  (dst)

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

Description: Performs "Inclusive OR" transfer 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 original contents of the destination are lost. The source is not affected.

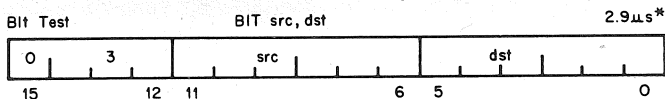


Operation:  $\sim$  (src)  $\wedge$  (dst)  $\rightarrow$  (dst)

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

Description: The BIC instruction 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 sources are unaffected.

\*There is no read/modify/write cycle in the CMP, BIT, and TST operations. This saves 0.4  $\mu$ s in all destination address modes except register mode.



Operation:  $(src) \wedge (dst)$

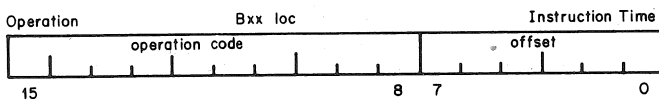
Condition Codes: Z: set if result = 0; cleared otherwise  
 N: set if high-order bit of result set; cleared otherwise  
 C: not affected  
 V: cleared

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.

Note that the operations of BIS, BIC, and BIT are parallel in that the same mask may be used to set, clear and test the state of particular bits in a word.

**BRANCHES**—Branches have the instruction format



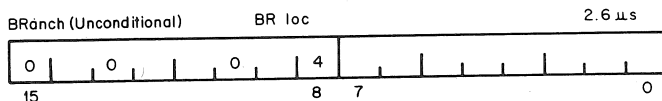
The offset is treated as a signed two's complement displacement to be multiplied by 2 and applied to the program counter. The program counter points to the next word in sequence. The effect is to cause the next instruction to be taken from an address, "loc", located up to 127 words back (— 254 bytes) or 128 words ahead (+ 256 bytes) of the branch instruction. PAL-11 gives an error indication in the instruction if "loc" is outside this range.

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 loc is the address to which the branch is to be made. The branch instructions have no effect on condition codes.

**Unconditional Branch—**

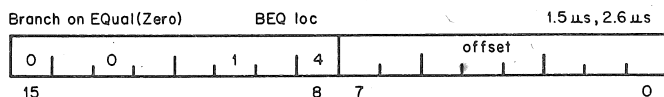


Operation:  $loc \rightarrow (PC)$

Description: Provides a way of transferring program control within a limited range with a one word instruction. The execution time is equal to the instruction time (2.6 μs) for the operation.

**Simple Conditional Branches**—Conditioned branches combine in one instruction a conditional skip, unconditional branch sequence.

Timing for the conditional branches is shown as execution time if the condition is not met, followed by the execution time if the condition is met (and a program branch occurs).



Operation: loc  $\rightarrow$  (PC) if Z = 1

Description: Tests the state of the Z-bit and causes a branch if Z is set. 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.

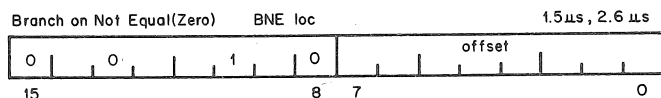
Thus the sequence

```
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

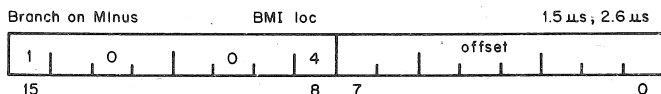
```
ADD  A,B      ; add A to B
BEQ  C        ; branch if the result = 0
```

will branch to C if A + B = 0.



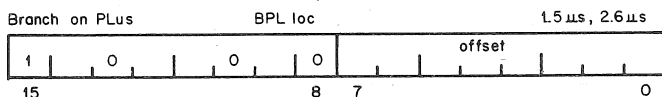
Operation: loc  $\rightarrow$  (PC) if Z = 0

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 set in the source, following a BIT and, generally, to test that the result of the previous operation was not zero.



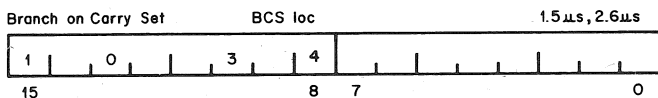
Operation: loc  $\rightarrow$  (PC) if N = 1

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.



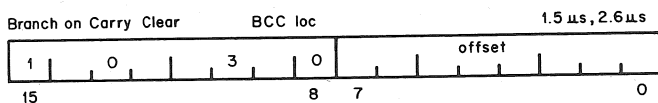
Operation:  $\text{loc} \rightarrow (\text{PC})$  if  $N = 0$ .

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



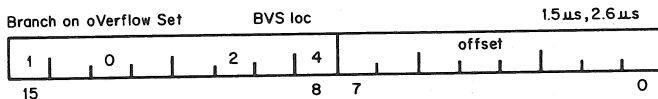
Operation:  $\text{loc} \rightarrow (\text{PC})$  if  $C = 1$

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.



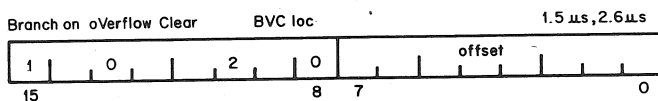
Operation:  $\text{loc} \rightarrow (\text{PC})$  if  $C = 0$

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



Operation:  $\text{loc} \rightarrow (\text{PC})$  if  $V = 1$

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



Operation:  $\text{loc} \rightarrow (\text{PC})$  if  $V = 0$

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

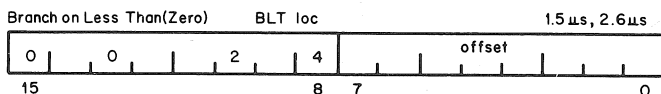
**Signed Conditional Branches**—Particular combinations of the condition code bits are tested with the signed conditioned branches. These instructions are used to test the results of instructions in which the operands were considered as signed (two's complement) values.

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  
 positive ..... 077776  
 .....  
 .....  
 ..... 000001  
 ..... 000000  
 ..... 177777  
 ..... 177776  
 .....  
 negative .....  
 .....  
 ..... 100001  
 smallest ..... 100000

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

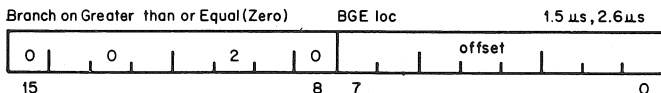
highest ..... 177777  
 .....  
 .....  
 .....  
 .....  
 ..... 000002  
 ..... 000001  
 lowest ..... 000000



Operation: loc  $\rightarrow$  (PC) if  $N \nabla V = 1$

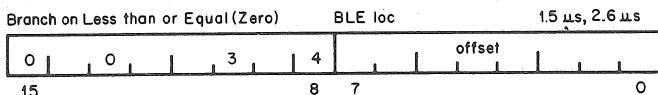
Description: Causes a branch if the "Exclusive OR" of the N- and V-bits are 1. Thus BLT will always branch following an operation that added two negative numbers, even if overflow occurred.

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).



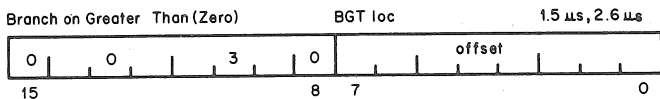
Operation: loc  $\rightarrow$  (PC) if  $N \nabla V = 0$

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 to two positive numbers. BGE will also cause a branch on a zero result.



Operation: loc  $\rightarrow$  (PC) if  $Z \vee (N \nabla V) = 1$

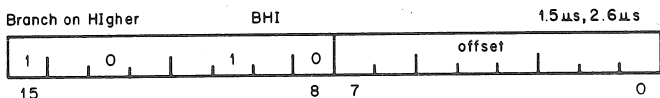
Description: Operation of BLE is similar to that of BLT but in addition will cause a branch if the result of the previous operation was zero.



Operation: loc  $\rightarrow$  (PC) if  $Z \vee (N \nabla V) = 0$

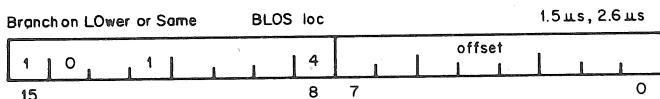
Description: Operation of BGT is similar to BGE, except that BGT will not cause a branch on a zero result.

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



Operation: loc  $\rightarrow$  (PC) if both C and Z = 0

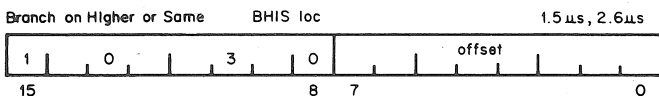
Description: Causes a branch if the previous operation caused neither a carry nor a zero result. This will happen in comparison (CMP) operations as long as the source has a higher unsigned value than the destination.



Operation: loc  $\rightarrow$  (PC) if  $C \vee Z = 1$

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.

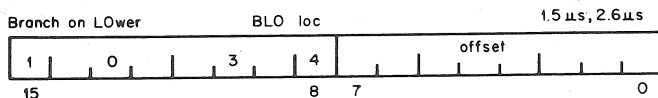
Comparison of unsigned values with the CMP instruction can be tested for "higher or same" and "higher" by a simple test of the C-bit. For convenience, the mnemonics BHIS (Branch on Higher or Same) and BLOS (Branch on Lower Or Same) have been defined such that BHIS = BCC and BLO = BCS.



Operation: loc  $\rightarrow$  (PC) if C = 0

Description: BHIS is the same instruction as BCC





Operation: loc  $\rightarrow$  (PC) if C = 1

Description: BLO is the same instruction as BCS

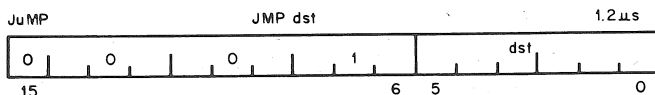
The following example illustrates the use of some of the instructions and addressing modes described thus far. Two new instructions are used: INC (INCrement) and ASL (Arithmetic Shift Left) which respectively, add 1 (INC) and multiply an operand by 2 (ASL). Their operation is fully described later in this chapter.

This example demonstrates the generation of a table (histogram) that shows the frequency of occurrence of each value in another table (within a range of values 1-100). Histogram generation (including initialization) requires 22 words. Values outside the range 1-100 are ignored.

```

HIST:      MOV #OTABLE, R0    ;set up to clear output table
           MOV #-100., R1     ;100 entries in output table
CLOOP:     CLR (R0)+          ;clear next entry
           INC R1             ;check if done
           BNE CLOOP          ;if not, continue clearing
           MOV #ITABLE, R0    ;set up input pointer
           MOV #-1000., R1    ;length of table
           MOV #100., R2      ;max input value
HLOOP:     MOV (R0)+, R4      ;get next input value
           BLE NOCOUNT       ;ignore if less than or equal zero
           CMP R4, R2          ;check against max value
           BGT NOCOUNT       ;ignore if greater
           ASL R4              ;2 bytes per table entry
           INC OTABLE (R4)    ;increment proper element
NOCOUNT:    INC R1            ;input done?
           BNE HLOOP          ;if not, continue scanning
           HALT               ;histogram complete
  
```

**The JuMP Instruction**—JMP (JuMP) provides more flexible program branching than is 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 PDP-11 addressing modes.



Operation: dst  $\rightarrow$  (PC)

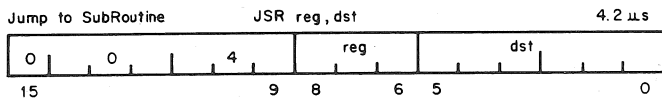
Conditioned Codes: not affected

Description: Register mode is illegal in JMP instructions and 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" 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.

**SUBROUTINES**—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 of return addresses at each level of subroutine call. The subroutine calling mechanism modifies no fixed location in memory and thus also provides for reentrancy. This allows one copy of a subroutine to be shared among several interrupting processes.



Operation:  $\text{dst} \rightarrow (\text{tmp})$  (tmp is an internal processor register)  
 $(\text{reg}) \downarrow$  (push reg contents onto processor stack)  
 $(\text{PC}) \rightarrow (\text{reg})$  (PC holds location following JSR; this address  
 $(\text{tmp}) \rightarrow (\text{PC})$  now put in reg)

Condition Codes: not affected

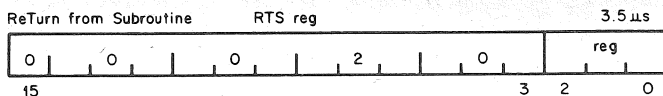
Description: Execution time for JSR is the sum of instruction and destination times. In execution of the JSR, the old contents of the specified register, (the "linkage pointer"), are automatically pushed onto the processor stack and new linkage information placed in the register. Thus subroutines nested 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.

A subroutine called with a JSR reg, dst instruction can access the arguments following the call with either autoincrement addressing,  $(\text{reg}) +$ , (if arguments are accessed sequentially) or by indexed addressing,  $\text{X}(\text{reg})$ , (if accessed in random order). These addressing modes may also be deferred,  $@(\text{reg}) +$  and  $@\text{X}(\text{reg})$  if the parameters are operand addresses rather than the operands 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. No register except the program counter is modified by this call.

Another special case of the JSR instruction is JSR PC,  $(\text{SP}) +$  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 the reg into the PC and pops the top element of the processor stack into the specified register.



Operation: (reg)  $\rightarrow$  (PC)  
 $\uparrow$  (reg)

Condition Codes: not affected

Description: Loads content of reg into PC and pops the top element of the processor stack into the specified register. Execution time for RTS is equal to the basic instruction time.

Return from a subroutine is typically made through the same register that was used in its call. Thus, a subroutine called with a JSR PC, dst exists with a RTS PC and a subroutine called with a JSR R5, dst, picks up parameters with addressing modes (R5)+, X(R5), or @X(R5) and finally exists with a RTS R5.

### Programming Examples of the Use of Subroutines—

1. Passing arguments in subroutine calls—The subroutine TOLER checks each element in an array of unsigned integers to determine whether any elements are outside specified limits. If all are within tolerance, the value 0 is returned in the register R0. If TOLER find an element out of tolerance, it returns the address of the bad element + 2 in R0. The calling sequence for TOLER is:

JSR R5, TOLER

- WORD ARRAY ;address of array to be  
;checked (.WORD expres-  
;sion—defines a word equal  
;to the value of the expres-  
;sion)
- WORD —LENGTH ;minus # of items in array
- WORD HILIM ;upper limit of tolerance
- WORD LOLIM ;lower limit of tolerance  
;subroutine returns here

;Tolerance Check-Array Elements Within Limits?

|         |               |                                    |
|---------|---------------|------------------------------------|
| TOLER:  | MOV (R5)+, R0 | ;get array address                 |
|         | MOV (R5)+, R1 | ;get minus the length              |
|         | MOV (R5)+, R2 | ;get high tolerance limit          |
|         | MOV (R5)+, R3 | ;get low tolerance limit           |
| TLOOP:  | MOV (R0)+, R4 | ;get next element of array         |
|         | CMP R4, R2    | ;check it against high limit       |
|         | BHI TEXTIT    | ;leave routine if higher           |
|         | CMP R4, R3    | ;check it against low limit        |
|         | BLO TEXTIT    | ;leave routine if lower            |
|         | INC R1        | ;increment count, check            |
|         |               | ;whether at end of array           |
|         | BNE TLOOP     | ;continue if not at end yet        |
|         | CLR R0        | ;exit with R0 = 0 if all ok        |
| TEXTIT: | RTS R5        | ;return, R0 holds pointer<br>;or 0 |

The instruction INC R1 increases the contents of R1 by 1 and the instruction CLR R0 zeroes the register R0

2. Saving and restoring registers on the stack—This subroutine pushes R0-R5 onto the stack. It is called by:

```

                JSR R5, SAVE
SAVE:           MOV R4, —(SP)      ;R5 was pushed by the JSR
                MOV R3, —(SP)      ;R5 will be at the bottom
                                ;of the stack
                MOV R2, —(SP)      ;R4, R3, R2, R1, and R0
                                ;in order
                MOV R1, —(SP)      ;will be above it
                MOV R0, —(SP)      ;R0 is at the top of the
                                ;stack
                JMP R5              ;R5 holds the return ad-
                                ;dress

```

The TST operation is equivalent to comparing the operand with 0, i.e.,

TST opr = CMP opr, #0

The only effect is to set the appropriate condition codes.

The following example illustrates a subroutine to restore R0-R5 from the stack.

```

REST:           TST (SP) +         ;this increments the SP by 2
                MOV (SP)+, R0      ;the registers are restored
                MOV (SP)+, R1      ;in reverse order to that in
                MOV (SP)+, R2      ;which
                MOV (SP)+, R3      ;they were put on the stack
                MOV (SP)+, R4      ;R5 is loaded into the PC
                RTS R5              and the old R5 restored

```

The operation TST (SP)+ removes the top element on the stack. At the time it is used, the top element holds the contents of R5 that were saved by the call to REST. Since R5 is to be loaded with the value saved on the stack by SAVE, this information is not needed.

3. Stacks, recursion, and nesting—The following subroutine converts an unsigned binary integer to a string of typed ASCII characters. In the routine, the remainders of successive divisions by 10 are saved and then typed in reverse order.

The operation of the subroutine is to call a part of itself (beginning with DECRM) repeatedly until a zero quotient is calculated by an integer divide subroutine, IDIVR. At each iteration, the dividend is divided by 10, the resulting quotient replaces the dividend, and the remainder is pushed onto the processor stack. The processor stack thus holds interleaved data (remainders) and control information (return addresses from calls to DECPNT and DECRM) when the quotient finally comes up as 0 and the branch is made to DECTTY. The portion of the routine beginning at DECTTY then pops a remainder from the stack, converts it to an ASCII character, types it and then returns control to DECTTY (with RTS PC) until the stack is reduced finally to its state immediately after the call to DECPNT.

At this point execution of RTS PC returns control to the main program.

A character is typed in DECTY by loading the teleprinter buffer (TPB) and waiting for the teleprinter READY flag, the most significant bit of the low-order byte of the teleprinter status word (TPS), to be set.

The symbols CR and LF are assumed equal to the ASCII representations for carriage return and line feed respectively.

This subroutine types the unsigned integer in R0. It illustrates recursion and the use of stacks.

|         |                |                               |
|---------|----------------|-------------------------------|
| DECPNT: | MOV #10., R2   | ;set up divisor of 10         |
| DECREM: | JSR PC, IDIVR  | ;subroutine divides (R0) by   |
|         |                | ;(R2)                         |
|         | MOV R1, —(SP)  | ;quotient is in R0, remain-   |
|         |                | ;der is in R1                 |
|         | TST R0         | ;after pushing remainder      |
|         |                | ;onto stack test quotient     |
|         | BEQ DECTTY     | ;if the quotient is 0, we're  |
|         |                | ;done getting remainders      |
|         | JSR PC, DECREM | ;if not try again             |
| DECTTY: | MOV (SP)+, R0  | ;get next remainder           |
|         | ADD #60, R0    | ;make an ASCII character      |
| TTYOUT: | MOV R0, TPB    | ;type the ASCII character in  |
|         |                | ;R0                           |
| TTYLUP: | TST TPS        | ;wait for the teleprinter to  |
|         |                | ;be done                      |
|         | BPL TTYLUP     | ;TPS is negative when the     |
|         |                | ;TP is done                   |
|         | CMP #CR, R0    | ;was the character of a car-  |
|         |                | ;riage return                 |
|         | BEQ TTYLF      | ;if not: return, if so; get a |
|         |                | ;line feed                    |
|         | RTS PC         | ;returns either to DECTTY     |
|         |                | ;or main program              |
| TTYLF:  | MOV #LF, TPB   | ;type a line feed             |
|         | BR TTYLUP      | ;and wait for it to be com-   |
|         |                | ;pleted                       |

4. Multiple entry points—In the example that follows, the subroutines described above are used to type out all the entries in a table of unsigned integers that are not within specified tolerance.

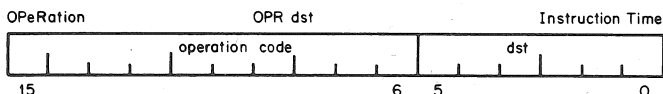
The subroutine TOLER is entered at TOLER for initialization and at TLOOP to pick up each bad entry of the array after the first one.

The subroutine DECPNT is entered at DECPNT to print the value of the unsigned binary number held in R0 and at TTYOUT to print the ASCII character held in R0. TTYOUT prints the carriage return, line feed sequence when it sees the carriage return character.

This routine types all out-of-tolerance elements of an integer array. The program starts at TYPOUT.

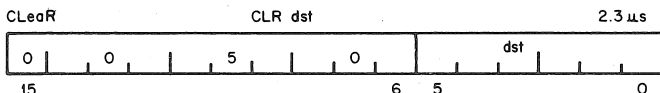
|         |                |   |
|---------|----------------|---|
| TYPFIN: | HALT           | ;suspend processor operation, wait for key continue |
| TYPOUT: | JSR R5, TOLER  | ;get address of bad item; initialization entry      |
|         | · WORD ARRAY   | ;address of array                                   |
|         | · WORD —LENGTH | ;length of array                                    |
|         | · WORD HILIM   | ;high limit   |
|         | · WORD LOLIM   | ;low limit  |
| TYPCHK: | BEQ TYPFIN     | ;Z-bit is set if no more out of limits              |
|         | JSR R5, SAVE   | ;an element is out of limits, save registers        |
|         | MOV —(R0), R0  | ;R0 holds address + 2, get operand into R0          |
|         | JSR PC, DECPNT | ;print out number                                   |
|         | MOV #CR, R0    | ;type CR, LF  |
|         | JSR PC, TTYOUT | ;note use of second entry point                     |
|         | JSR R5, REST   | ;restore registers                                  |
|         | JSR R5, TLOOP  | ;continue searching array, alternate entry          |
|         | BR TYPCHK      | ;another bad element?                               |

**SINGLE OPERAND INSTRUCTIONS**—Single Operand Instructions are represented as:



The execution time for single operand instructions is the sum of the basic instruction time and destination address time for the operation.

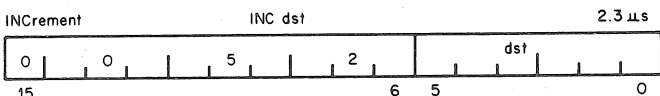
### General Operations—



Operation: 0 → (dst)

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

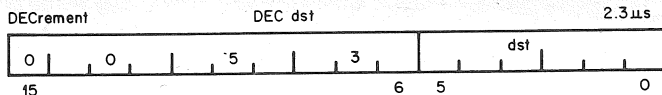
Description: Zeroes the specified destination.



Operation: (dst) + 1 → (dst)

Condition Codes: Z: set if the result is 0; cleared otherwise  
N: set if the result is < 0; cleared otherwise  
C: not affected  
V: set if (dst) held 077777; cleared otherwise

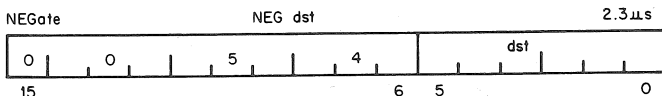
Description: Adds 1 to the contents of the destination.



Operation: (dst) — 1 → (dst)

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

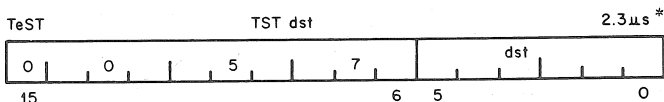
Description: Subtracts 1 from the contents of the destination.



Operation: — (dst) → (dst)

Condition Codes: as in SUB dst, #0  
                          Z: set if the result is 0; cleared otherwise  
                          N: set if the result is < 0; cleared otherwise  
                          C: cleared if the result is 0; set otherwise  
                          V: set if the result is 100000; cleared otherwise

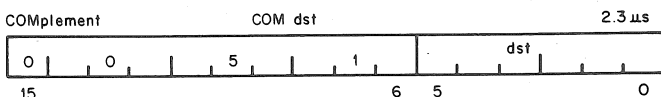
Description: Replaces the contents of the destination address by their two's complement. (However, 100000<sub>2</sub> is replaced by itself—in two's complement notation the most negative number has no positive counterpart.)



Operation: 0 — (dst)

Condition Codes: as in CMP #0, dst  
                          Z: set if the result is 0; cleared otherwise  
                          N: set if the result is < 0; cleared otherwise  
                          C: cleared  
                          V: cleared

Description: Sets the condition codes Z and N according to the contents of the destination address.



Operation: ~ (dst) → (dst)

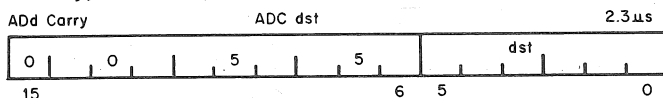
Condition Codes:     Z: set if result is 0; cleared otherwise  
                          N: set if most significant bit of result set; cleared otherwise  
                          C: set  
                          V: cleared

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).

\* See the note for the CMP instruction.



**Multiple Precision Operations**—It is sometimes convenient to do arithmetic on operands considered as multiple words. The PDP-11 makes special provision for such operations with the instructions ADC (ADD Carry) and SBC (SuBtract Carry).



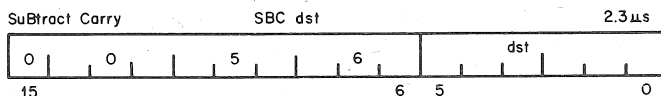
Operation:  $(dst) + (C) \rightarrow (dst)$

Condition Codes: Z: set if result = 0; cleared otherwise  
 N: set if result < 0; cleared otherwise  
 C: set if (dst) was 177777 and (C) was 1; cleared otherwise  
 V: set if (dst) was 077777 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 two low-order words to be carried into the high-order result.

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
```



Operation:  $(dst) - (C) \rightarrow (dst)$

Condition Codes: Z: set if the result 0; cleared otherwise  
 N: set if the result < 0; cleared otherwise  
 C: cleared if the result is 0 and C = 1; set otherwise  
 V: set if the result is 100000; cleared otherwise

Description: 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.

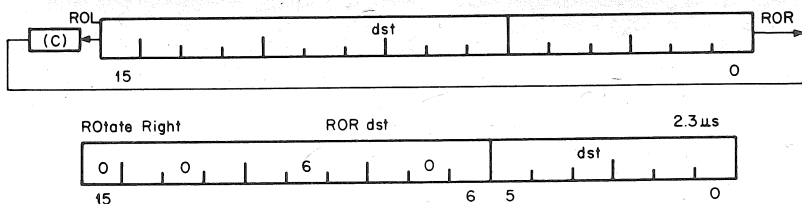
Double precision subtraction is done by:

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

Double precision negation is accomplished with:

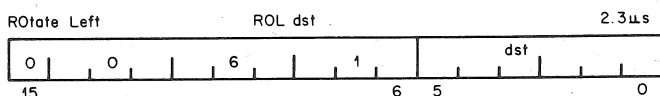
```
NEG B0 ;negate low-order part; sets C unless B0 = 0
SBC B1 ;makes "NEG B1" = "COMB B1" unless B0 = 0
NEG B1 ;negate high-order part
```

**Rotates**—Testing of sequential bits of a word and detailed bit manipulation are aided with rotate operations. The instructions ROR (ROTate Right) and ROL (ROTate Left) cause the C-bit of the status register to be effectively appended to the destination operand in circular bit shifting.



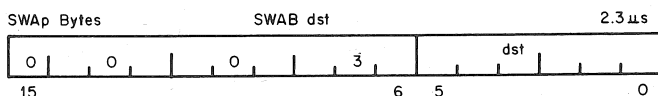
Condition Codes: Z: set if all bits of result = 0; cleared otherwise.  
 N: set if the high-order bit of the result is set; cleared otherwise  
 C: loaded with the low-order bit of the destination  
 V: loaded with the Exclusive OR of the N-bit and C-bit (as set by the completion of the rotate operation).

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



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

Description: Rotates 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.



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

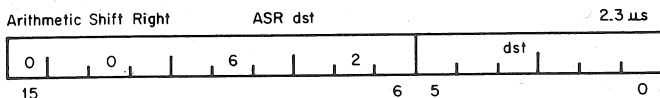
Description: Exchanges high-order byte and low-order byte of the destination word (dst must be a word address).

**Shifts**—Scaling data by factors of 2 is accomplished by the shift instructions:

ASR—Arithmetic Shift Right

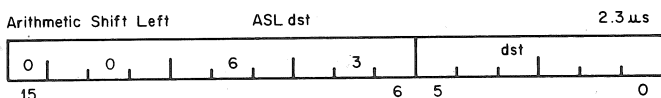
ASL—Arithmetic Shift Left

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 are lost.



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

Description: 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 2.



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

Description: Shifts all bits of the destination left one place. Bit 0 is loaded with a 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.

Multiple precision shifting is done with a sequence of shifts and rotates.

#### Double Precision Right Shift:

ASR A1; low-order bit of A1 to C-bit  
 ROR A0; C-bit to high-order bit of A0

#### Double Precision Left Shift:

ASL A0; high-order bit of A0 to C-bit  
 ROL A1; C-bit to low-order bit of A1

Normalization of operands (scaling of the operand until the operand taken as a 15-bit fraction with sign is in the range  $-\frac{1}{2} < \text{operand} \leq \frac{1}{2}$ ) proceeds as follows:

|        |     |       |  |
|--------|-----|-------|--|
| NORM:  | ASL | A     | ; shift 0's into low-order bit         |
|        | BEQ | NFIN  | ; if the result is 0, the operation is |
|        |     |       | ; complete                             |
|        | BVC | NORM  | ; if the sign did not change, continue |
|        | ROR | A     | ; restore the sign                     |
|        | BR  | NDONE | ; normalization complete               |
| NFIN:  | ROR | A     | ; restore the sign: 000000 or 100000   |
|        | ASR | A     | ; and replicate it: 000000 or 140000   |
| NDONE: |     | ...   |  |

Double precision normalization proceeds similarly:

```

DNORM:    ASL    A0      ; double precision left shift
          ROL    A1      ;
          BEQ    DZERO   ; high order result 0?. if so, check low
          BVC    DNORM   ; if the sign did not change, continue
          ROR    A1      ; restore the sign
          BR     DNDONE   ; normalization complete
DZERO:    TST    A0      ; low order zero, too?
          BNE    DNORM   ; if not, continue normalization
          ROR    A1      ; restore the sign; 000000 or 100000
          ASR    A1      ; and replicate it; 000000 or 140000
DNDONE:    . . .

```

The following example illustrates the use of shifts and rotates in a 16-bit unsigned integer multiply subroutine. Access of operands through address parameters following the subroutine is also shown. The multiplication takes 115-170  $\mu$ s in in-line code. The entire subroutine as shown below takes approximately 200  $\mu$ s and requires 16 words. The calling sequence is JSR R5, MULT.

```

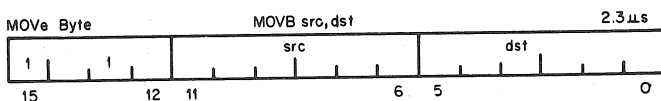
      . WORD MCAND      ; address of multiplicand
      . WORD MPLIER     ; address of multiplier
      . WORD PROD       ; address of product
MULT:  CLR R0
      MOV @ (R5) +, R1  ; get multiplier into R1
      MOV @ (R5) +, R2  ; get multiplicand into R2
      MOV #-16., R3     ; set counter
MLOOP: ASL R0           ; double prec shift
      ROL R1           ; shift and add multiply
      BCC NOADD        ; most significant bit governs add
      ADD R2, R0       ; if set add in multiplicand
      ADC R1           ; keep 32-bit product
NOADD: INC R3          ; done?
      BNE MLOOP        ; if not continue
      MOV (R5) +, R2    ; get address to store prod.
      MOV R0, (R2) +    ; put low-order away, move to high
      MOV R1, (R2)      ; put high-order away
      RTS R5           ; return to calling program

```

**BYTE OPERATIONS**—The PDP-11 processor includes a full complement of instructions that manipulate byte operands. Addressing is byte-oriented so that instructions for byte manipulation are straightforward. In addition, byte instructions with autoincrement or autodecrement direct addressing cause the specified register to be stepped 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.

Timing of byte instructions is the same as for word instructions except that an additional 0.6  $\mu$ s is required for access of bytes at odd addresses.

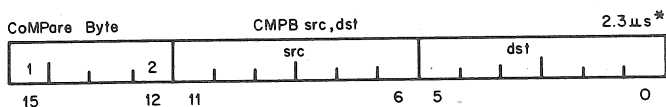
#### Double Operand Byte Instructions—



Operation: (src)  $\rightarrow$  (dst)

Condition Codes: Set on the byte result as in MOV

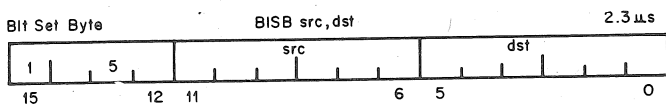
Description: Same as MOV instruction. The MOV<sub>B</sub> instruction in register mode (unique among byte operations) extends the most significant bit of the byte register (sign extension). Otherwise MOV<sub>B</sub> operates on bytes exactly as MOV operates on words.



Operation: (src)  $-$  (dst) ; in detail (src)  $+ \sim$  (dst)  $+ 1$

Condition Codes: Set on the byte result as in CMP

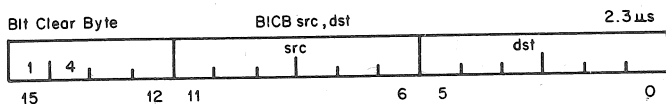
Description: Same as CMP instruction.



Operation: (src)  $\vee$  (dst)  $\rightarrow$  (dst)

Condition Codes: Set on the byte result as in BIS

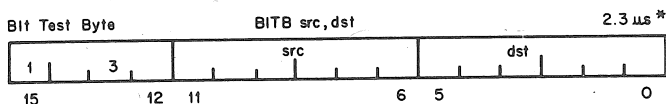
Description: Same as BIS.



Operation:  $\sim$  (src) (dst)  $\rightarrow$  (dst)

Condition Codes: set on the byte result as in BIC

Description: Same as BIC.



Operation: (src) (dst)

Condition Codes: Set on the byte result as in BIT

Description: Same as BIT.

The following subroutine scans a packed character string of variable length lines, removes blanks and unpacks the string to left-justified length lines. INSTRING is the address of the INPUT STRING, OUTSTRING is the address of the OUTPUT String. EOLCHAR, SPCHAR, and EORCHAR are the End Of Line CHARACTER, SPace CHARACTER, and End of Record CHARACTER respectively.

\* See the note for the CMP instruction.

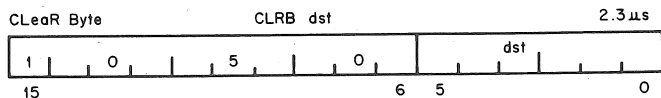
LNLINE is the Length of uNpacked LINES. The routine requires 24 words.

```

EDIT:      MOV #INSTRING, R0      ; set up input byte pointer
           MOV #OUTSTRING, R1     ; set up output byte pointer
           MOV #EOLCHAR, R2       ; put high use constant in reg.
           MOV #SPCHAR, R3        ; to save time in loop
NOLINE:    MOV #LNLINE, R4        ; R4 holds # char left in line
NXTCHR:    MOVB (R0) + ,R5        ; get next byte
           CMP R5, R2              ; end of line?
           BEQ FILINE              ; if yes, fill line
           CMP R5, R3              ; blank?
           BEQ NXTCHR              ; if yes, skip character
           DEC R4                  ; decrement # of characters left in line
           MOVB R5, (R1) +        ; move byte to output string
           BR NXTCHR               ; continue
FILINE:    CLRB (R1) +            ; put a blank byte in output
           DEC R4                  ; decrement # char left
           BNE FILINE              ; continue if not end
CHKEND:    CMPB (R0), #EORCHAR    ; end of record?
           BNE NULINE              ; if not EOR, start next line

```

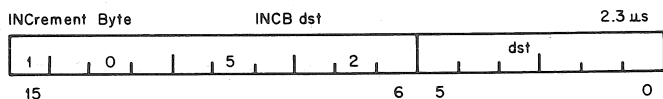
### Single Operand Byte Instructions—



Operation: 0  $\rightarrow$  (dst)

Condition Codes: Set on the byte result as in CLR

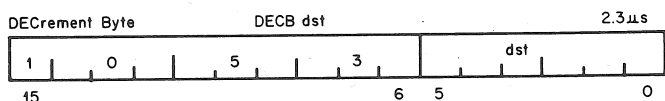
Description: Same as CLR



Operation: (dst) + 1  $\rightarrow$  (dst)

Condition Codes: Set on the byte result as in INC

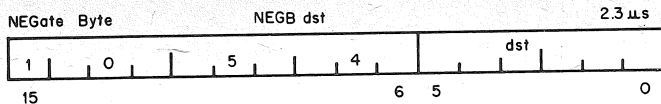
Description: Same as INC. The carry from a byte does not affect any other byte.



Operation: (dst) - 1  $\rightarrow$  (dst)

Condition Codes: Set on the byte result as in DEC

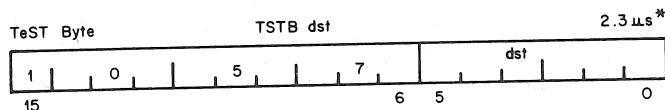
Description: Same as DEC.



Operation:  $\sim(\text{dst}) \rightarrow (\text{dst})$  ; in detail,  $\sim(\text{dst}) + 1 \rightarrow (\text{dst})$

Condition Codes: Set on the byte result as NEG

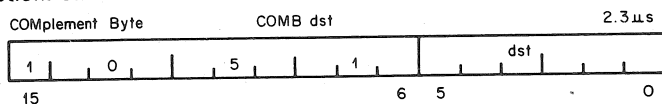
Description: Same as NEG.



Operation:  $0 - (\text{dst})$

Condition Codes: Set on the byte result as TST

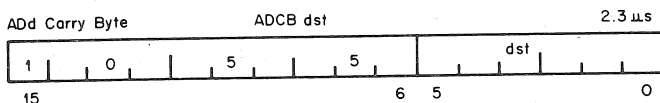
Description: Same as TST.



Operation:  $\sim(\text{dst}) \rightarrow (\text{dst})$

Condition Codes: Set on the byte result as COM

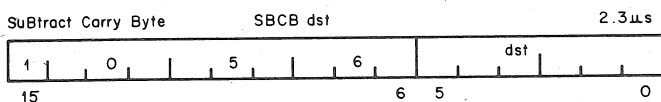
Description: Same as COM.



Operation:  $(\text{dst}) + (\text{C}) \rightarrow (\text{dst})$

Condition Codes: Set on the byte result as ADC

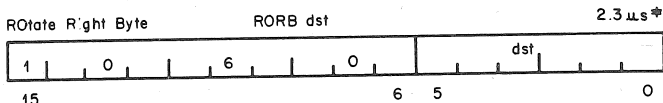
Description: Same as ADC.



Operation:  $(\text{dst}) - (\text{C}) \rightarrow (\text{dst})$

Condition Codes: Set on the byte result as SBC

Description: Same as SBC.

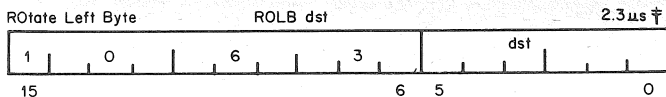


Operation: as in ROR on byte operands

Condition Codes: Set on the byte result as ROR

Description: Same as ROR

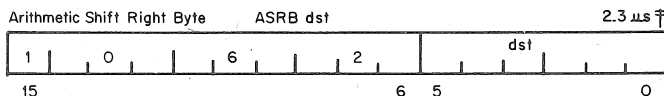
\* See the note for the CMP instruction.



Operation: as in ROL on byte operands

Condition Codes: set on the byte results as ROL

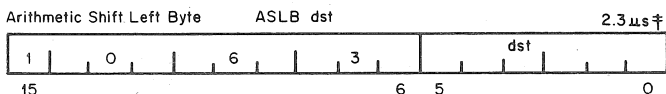
Description: Same as ROL



Operation: as in ASR on byte operands

Condition Codes: set on the byte result as ASR

Description: Same as ASR

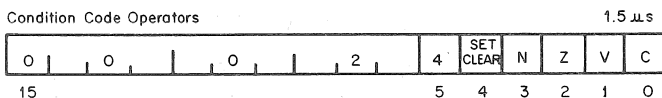


Operation: as in ASL on byte operands

Condition Codes: set on the byte results as ASL

Description: Same as ASL

**CONDITION CODE OPERATORS**—Condition code operators set and clear condition code bits. Selectable combinations of these bits may be cleared or set together in one instruction.



Condition code bits corresponding to bits in the condition code operator (bits 3-0; N, Z, V, C) are modified according to the sense of bit 4, the set/clear bit of the operator. The following mnemonics are permanent symbols in the assembler:

| Mnemonic | Operation | Op Code | Mnemonic | Operation | Op Code |
|----------|-----------|---------|----------|-----------|---------|
| CLC      | Clear C   | 000241  | SEC      | Set C     | 000261  |
| CLV      | Clear V   | 000242  | SEV      | Set V     | 000262  |
| CLZ      | Clear Z   | 000244  | SEZ      | Set Z     | 000264  |
| CLN      | Clear N   | 000250  | SEN      | Set N     | 000270  |

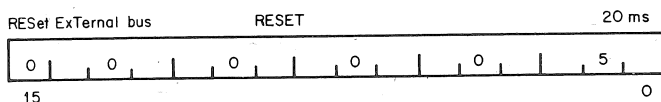
Timing for all condition code operators is the basic instruction time (1.5  $\mu$ s) for the operators. (The codes 000240 and 000260 are the shortest "no-operation" instructions.)

† Shift and rotate operations require an additional 0.6  $\mu$ s to access bytes at odd addresses.



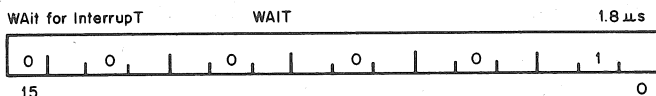
Combinations of the above set or clear operations may be ORed together to form new instruction mnemonics. For example:  $CLCV = CLC \text{ ! } CLV$ . The new instruction clears C and V bits. ("!" signifies "inclusive or" in PAL-11.)

## MISCELLANEOUS CONTROL INSTRUCTIONS



Condition Codes: not affected

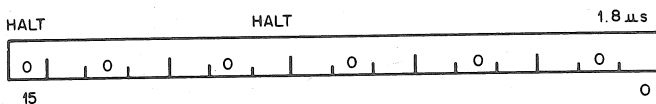
Description: Sends an INIT pulse along the Unibus by the processor. All devices on the bus are reset to their state at power-up.



Condition Codes: not affected

Description: Provides a way for the processor to relinquish use of the bus while it waits for an external 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 an interrupt causes the PC and PS to be pushed onto the processor stack, the address of the next instruction following the WAIT is saved. The exit from the interrupt routine (i.e. execution of an RTI instruction) will cause resumption of the interrupted process at the instruction following the WAIT.

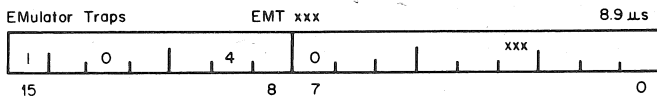


Condition Codes: not affected

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 of 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.

**Processor Traps** —Processor Traps are internally generated interrupts. Error conditions, completion of an instruction in trace mode (i.e. T-bit of status word set), and certain instructions cause traps. As in interrupts, the current PC and PS are saved on the processor stack and a new PC and PS are loaded from the appropriate trap (interrupt) vector. See Appendix C for a summary of Trap Vector Addresses.

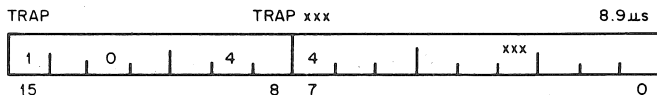
**Trap Instructions**—Trap Instructions provide for calls to emulators, I/O monitors, debugging packages, and user-defined interpreters.



Operation: (PS)  $\downarrow$  SP  
 (PC)  $\downarrow$  SP  
 (30)  $\rightarrow$  PC  
 (32)  $\rightarrow$  PS

Condition Codes: loaded from trap vector.

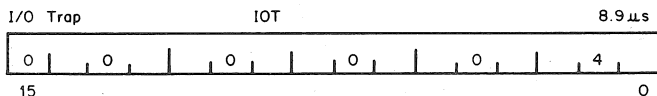
Description: Performs a trap sequence with a trap vector address of 30. All operation codes from 104000 to 104377 are EMT calls. The low-order byte, bits 0-7 of the EMT instructions, may be used to transmit information to the emulating routine (e.g., function to be performed). The trap vector for EMT 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.



Operation: as in EMT except the trap vector is located at 34.

Condition Codes: loaded from trap vector.

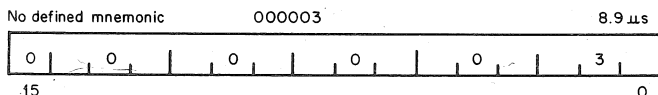
Description: Performs a trap sequence with a trap vector address of 34. Operation codes from 104400 to 104777 are TRAP instructions. TRAPs and EMTs are identical in operation, except that the trap vector for TRAP is at address 34.



Operation: as EMT except the trap vector is located at address 20 and no information is transmitted in the low byte.

Condition Codes: loaded from trap vector.

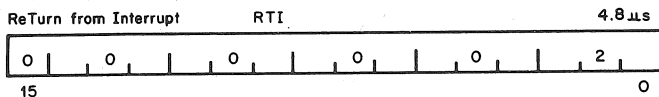
Description: Used to call the I/O executive routine IOX.



Operation: Same as IOT except that trap vector is located at address 14.

Condition Codes: loaded from trap vector.

Description: Used to call debugging aids. The user is cautioned against employing code 000003 in programs run under these debugging aids.



Operation: SP ↑ (PC), SP ↑ (PS).

Condition Codes: 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.

Instruction traps are also 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 above are considered to be reserved instructions. Illegal instructions are JMP and JSR with register mode destinations. Reserved and illegal instruction traps occur as described under EMT, but trap through vectors at addresses 10 and 04 respectively.

**Stack Overflow Trap**—Stack Overflow Trap is a processor trap through the vector at address 4. It is caused by referencing addresses below 400, through the processor stack pointer R6 (SP) in autodecrement or autodecrement deferred addressing. The instruction causing the overflow is completed before the trap is made.

**Bus Error Traps**—Bus Error Traps are:

1. Boundary Errors—attempts to reference word operands at odd addresses.
2. Time-Out Errors—attempts to reference addresses on the bus that made no response within 10 μs. In general, these are caused by attempts to reference nonexistent memory, and attempts to reference nonexistent peripheral devices.

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

**Trace Trap**—Trace Trap enables bit 4 of the PS word 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.

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.

**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.

*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 exists 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*—This is treated as in 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 no interrupts are acknowledged between the time of fetching any trapped instruction (including one that is trapped by reason of the T-bit being set) and completing execution of the first instruction of the trap service.

*A WAIT*—The trap occurred immediately. The address of the next instruction is saved on the stack.

*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.

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

1. Bus Errors
2. Instruction Traps
3. Trace Trap
4. Stack Overflow 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.

# CHAPTER 5

## ADDRESS ALLOCATION

The PDP-11 provides for a very flexible addressing structure. Both 16-bit words and 8-bit bytes can be directly addressed. Addresses are 16-bits long allowing for direct addressing of 32,768 words or 65,536 bytes.

### ADDRESS MAP

As a result of the organization of the PDP-11, bus addresses serve several functions. A map of possible PDP-11 bus address allocation is shown

| BUS ADDRESS         | CONTENT                          |               |
|---------------------|----------------------------------|---------------|
| 0                   |                                  |               |
| .                   | Program Counter                  | Processor     |
| .                   |                                  | Trap Vectors  |
| .                   | Processor Status Word            | and Device    |
| .                   |                                  | Interrupt     |
| .                   |                                  | Vectors       |
| .                   |                                  |               |
| 400 <sub>8</sub>    | Stack Pointer Overflow Limit     |               |
| .                   | Stacks, Program and Data Storage |               |
| .                   |                                  |               |
| .                   |                                  |               |
| 160000 <sub>8</sub> |                                  |               |
| .                   |                                  | Typical       |
| .                   |                                  | Registers for |
| .                   | Status Register and              | Programmed    |
| .                   | Data Buffer Register             | Transfer      |
| .                   |                                  | Device        |
| .                   |                                  |               |
| .                   |                                  |               |
| .                   | Device Address Register          | Typical       |
| .                   | Word Count Register              | Registers     |
| .                   | Memory Address Register          | for a         |
| .                   | Control and Status Registers     | Block         |
| .                   |                                  | Transfer      |
| .                   |                                  | Device        |
| .                   |                                  |               |
| 177777 <sub>8</sub> |                                  |               |

Figure 5-1  
Simplified Address Allocation Map

in Figure 5-1. Three areas of addresses of particular interest to the programmers are: 1) Interrupt and Trap Vectors; 2) Processor Stack and General Storage; and 3) Peripheral Device Registers.

**INTERRUPT AND TRAP VECTORS**—Addresses between location zero and location 400, are generally reserved for interrupt and trap vectors.

**PROCESSOR STACK AND GENERAL STORAGE**—Addresses between 400, and the limit of implemented core are available for the processor stack or other programs and data. The highest address in this region is 157777.

**PERIPHERAL DEVICE REGISTERS**—Addresses above 160000, generally are reserved for peripheral device status, control, and data registers. The general registers and the processor status can be addressed from the program console using addresses in this area.

A more detailed address allocation map can be found in Appendix D.

## **CORE MEMORY**

The three types of core memory that can be used in a PDP-11 system are: 1) Read-Write Core Memory; 2) Read-Only Core Memory; and 3) Wordlet Memory. These memories can be located anywhere in address space provided they do not overlap. They do not have to be in continuous address locations.

**MM11-E READ WRITE CORE MEMORY**—The MM11-E has the following specifications:

Capacity: 4,096 16-bit words or 8,192 8-bit bytes

Cycle Time: 1.2 microseconds

Access Time: 500 nanoseconds

Configuration: Planar 3-wire, 3-D using 22 mil cores

Packaging: One standard PDP-11 System Unit

Interface: Designed to work with PDP-11 bus, TTL-compatible

**MR11-A READ-ONLY CORE MEMORY (ROM)**—The ROM has the following specifications:

Capacity: 1,024 16-bit words or 2,048 8-bit bytes

Access Time: 500 nanoseconds

Configuration: 2-piece core with wire braid, 256 wires, 64 cores

Packaging: 3/4 of one standard PDP-11 System Unit

Interface: Designed to work with PDP-11 bus, TTL-compatible

**MW11-A WORDLET MEMORY**—The wordlet memory is used with ROM systems and provides read-write memory capacity for temporary data and instruction storage.

Capacity: 128 16-bit words or 256 8-bit bytes

Cycle Time: 2.0 microseconds

Access Time: 1.0 microsecond

Configuration: 5-Wire, 3D

Packaging: 1/4 standard PDP-11 single System Unit

Interface: The wordlet memory will work with the ROM and interfaces through the ROM System Unit to the PDP-11 bus.

## CHAPTER 6

### PROGRAMMING OF PERIPHERALS

Programming of peripherals is extremely simple in the PDP-11—a special class of instructions to deal with input/output operations is unnecessary. The Unibus permits a unified addressing structure in which control, status, and data registers for peripheral devices are directly addressed as memory locations. Therefore all operations on these registers, such as transferring information into or out of them or manipulating data within them, are performed by normal memory reference instruction.

The ability to use all memory reference instructions on peripheral device registers greatly increases the flexibility of input/output programming. Information in a device register can be compared directly with a value and a branch made on the result.

CMP #101, PRB  
BEQ SERVICE

In this case the program looks for 101<sub>8</sub> from the paper tape reader data buffer, and branches if it finds it. There is no need to transfer the information into an intermediate register for comparison.

When the character is of interest, a memory reference instruction can transfer the character into a user buffer in core or in another peripheral device.

MOV PRB, LOC

This instruction transfers a character from the paper tape reader buffer into a user-defined location.

All arithmetic operations can be performed on a peripheral device register.

ADD #10, DSX

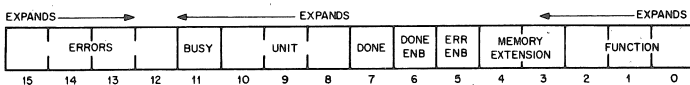
This instruction will add 10<sub>8</sub> to a display's x-deflection register.

All peripheral device registers can be treated as accumulators. There is no need to funnel all data transfers, arithmetic operations, and comparisons through a single or small number of accumulator registers.

### DEVICE REGISTERS

All peripheral devices are specified by a set of registers which are addressed as core memory and manipulated as flexibly as an accumulator. There are two types of registers associated with each device: 1) Control and Status Registers (CSR); and 2) Data Registers.

**CONTROL AND STATUS REGISTERS (CSR)**—Each peripheral has one or more control and status registers which contain all the information necessary to communicate with that device. The general form of a control and status register is shown below.



General Control and Status Register

This general form does not necessarily apply to any device, but is presented as a format which could be used as a guideline for designing peripheral





acter. Incrementing the PRS will set bit 0 and cause one character to be read. The instruction

INC     PRS

performs that function. MOV #1, PRS does the same thing but takes one more word.

**DATA BUFFER REGISTERS**—Each device has at least one buffer register for temporarily storing data to be transfer into or out of the computer. The number and type of data registers is a function of the device. The paper tape reader and punch use single 8-bit data buffer registers. A disk would use 16-bit data registers and some devices may use two 16-bit registers for data buffers.

## PROGRAMMING EXAMPLES

### PROGRAM CONTROLLED DATA TRANSFER WITH THE INTERRUPT DISABLED

—Single character I/O devices (teletype, paper tape reader/punch) have an addressable register buffer through which data is transferred. For input, the data buffer register is the source operand of the instruction used to get the data; for output, it is the destination operand. For example assuming the paper tape reader interrupt is not enabled, character input could proceed as follows:

```

      MOV R, —(SP)           ; save R on the stack
      MOV #BUFFER, R         ; pointer to input buffer into register R
START: INC PRS                ; start up reader
LOOP:  BIT PRS, #100200      ; test DONE and ERROR bits
      BEQ LOOP               ; branch back if none on yet
      BMI ERROR              ; branch to error routine if minus
      MOVB PRB, (R)+         ; move byte from device buffer reg-
                              ; ister to user's buffer and increment
                              ; pointer
      CMP #LIMIT R,         ; check for end of buffer
      BGE START              ; get next character
      MOV (SP)+, R           ; restore R

```

Character output to the paper tape punch might be executed as follows:

```

      MOV R0, —(SP)          ; save R0
      MOV R1, —(SP)          ; save R1
      MOV NCHAR, R0           ; number of characters into R0
      MOV BUFFER, R1          ; user buffer address into R1
LOOP:  BIT PPS, #100200      ; test device ready and error bits
      BEQ LOOP               ; fall through if on
      BMI ERROR              ; branch on error
      MOVB (R1)+, PPB         ; output character, increment pointer
      DEC R0                  ; decrement character counter (and
                              ; set condition codes)
      BGT LOOP               ; repeat if greater than zero
      MOV (SP)+, R0           ; restore R0
      MOV (SP)+, R1           ; restore R1

```

**BLOCK TRANSFER WITH THE INTERRUPT DISABLED**—High-speed block transfer devices use the Unibus to make data transfers between the device and core memory. These devices are provided with addressable registers that control the flow of data.

A typical set might be:

1. Control and status register
2. Memory address register
3. Word count register
4. Device address register

Loading the device address register would in general initiate the transfer, which then proceeds without processor intervention. The device issues non-processor requests for the Unibus that, when granted, allow direct data transfer between the device and memory. These requests are interleaved with processor requests for the bus. If very fast transfer is required, the processor may execute a WAIT instruction after starting the block transfer.

The DONE or appropriate error bits are set in the CSR with completion of the transfer or when an error occurs. These may be enabled to cause an interrupt or may be tested to determine when the device needs assistance.

A block transfer could be executed as follows:

```
MOV  #401, DKS,           ; read block of data (function 1)
                                ; from unit 1
MOV  #BUFADR, DKMA        ; buffer address to memory ad-
                                ; dress register
MOV  #BUFCNT, DKWC        ; word count to word count register
MOV  #BLKNO, DKDA         ; block number to device address
                                ; register, which starts the trans-
                                ; fer
.
.
.
; when data is needed.

LOOP: BIT  #DKMSK, DKS    ; test done bit and error bits
      BEQ  LOOP          ; branch back if none on
      BIT  #DKEMSK, DKS   ; test for any error bits
      BNE  ERROR         ; branch if any on
; data is now in buffer at BUFADR
```

## INTERRUPT STRUCTURE

If the appropriate interrupt enable bit is on, in the control and status register of a device, transition from 0 to 1 of the DONE or READY bit causes an interrupt request to be issued to the processor. Also if DONE or READY is a 1 when the interrupt enable is turned on, an interrupt request is made. If the device makes the request at a priority greater than that at which the processor is running and no other conflicts exist, the request is granted and the interrupt sequence takes place:

- a. the current program counter and processor status are pushed onto the processor stack;
- b. the new PC and PS are loaded from a pair of locations (the interrupt vector) in low core unique to the interrupting device.

Since each device has a unique interrupt vector which dispatches control to the appropriate interrupt handling routine immediately, no device polling is required. Furthermore, since the PS contains the processor priority, the priority at which an interrupt request is serviced can be set under program control and is independent of the priority of the interrupt request. The

ReTurn from Interrupt instruction is used to reverse the action of the interrupt sequence. The top two words on the stack are popped into the PC and PS, returning control to the interrupted sequence.

## PROGRAMMING EXAMPLE

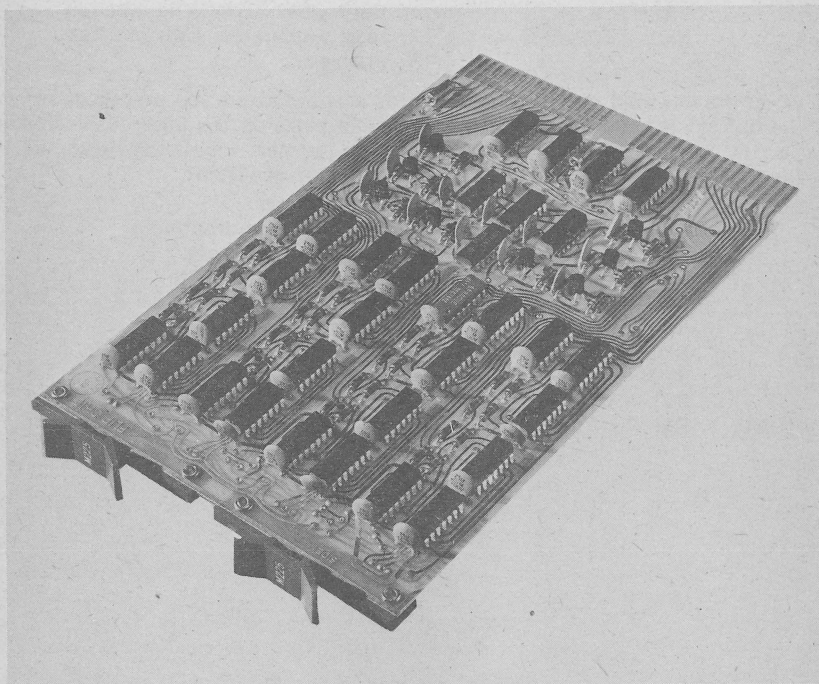
A paper tape reader interrupt service could appear as follows:

First the user must initialize the service routine by specifying an address pointer and a word count

```
INIT:      MOV #BUFADR, #0      ; set up address pointer
           POINTR = . - 2      ; in third word of MOV instruction.
           MOV #CNTR, #0       ; set up character count in
           CRCNT = . - 2       ; third word of MOV instruction.
           MOV #101, PRS       ; read a character with interrupt
                                   ; enabled.
```

When the interrupt request occurs and is acknowledged, the processor stores the current PC and PS on the stack. Next it picks up the interrupt vector or new PC and PS beginning at location 70<sub>h</sub>. The next instruction executed is the first instruction of the device service routine at PRSER.

```
PRSER:     TST     PRS          ; test for error
           BMI     ERROR       ; branch to error routine if
                                   ; bit 15 of PRS is set.
           MOVB    PRB, @POINTR ; move character (byte)
                                   ; from reader to buffer
           INC     POINTR      ; increment pointer
           DEC     CRCNT       ; decrement character count
           BEQ     DONE        ; branch when input done
           INC     PRS         ; start reader for next character
DONE:      RTI                ; return from interrupt
```



The DIGITAL M225 module contains 8 high speed general-purpose registers. The M225 general registers provide program flexibility when used as accumulators, index registers, and pointers to data words.

## CHAPTER 7

### TELETYPE (MODEL LT33-DC/DD)

The standard Teletype Model 33 ASR (Automatic Send-Receive) can be used to type in or print out information at a rate of up to ten characters per second, or to read in or punch out perforated paper tape at a ten characters per second rate. Signals transferred between the 33 ASR and the control logic are standard serial, 11-unit code Teletype signals. The signals consist of "marks" and "spaces" which correspond to idle and bias current in the Teletype serial line, and to 0's and 1's in the control and computer. The start mark and subsequent eight bits are each one unit of time duration and are followed by the stop mark which is two units.

The 8-bit code used by the Model 33 ASR Teletype unit is the Americal Standard Code for Information Interchange (ASCII) modified. To convert the ASCII code to Teletype code, add 200 octal ( $\text{ASCII} + 200_8 = \text{Teletype}$ ).

The Model 33 ASR can generate all assigned codes except 340 through 374 and 376. The Model 33 ASR can detect all characters, but does not interpret all codes that it can generate as commands. The standard number of characters printed per line is 72. The sequence for proceeding to the next line is a carriage return followed by a line feed. Punched tape format is as follows:

| Tape Channel                 | 87 | 654 | S | 321 |
|------------------------------|----|-----|---|-----|
| Binary Code<br>(Punch = 1)   | 10 | 110 |   | 100 |
| Octal Code<br>(S = Sprocket) | 2  | 6   |   | 4   |

**SIZE—** Floor space approximately  $22\frac{1}{4}$ " wide,  $18\frac{1}{2}$ " deep  
Cable length 8 feet

#### MODEL

#### POWER REQUIREMENTS

|         |                  |                  |
|---------|------------------|------------------|
| LT33-DC | 115 V $\pm 10\%$ | 60 $\pm 0.45$ Hz |
| LT33-DD | 230 V $\pm 10\%$ | 50 $\pm 0.75$ Hz |

### TELETYPE CONTROL (MODEL KL11)

**TELETYPE CONTROL—**Serial information read or written by a Teletype unit is assembled or disassembled by the control for parallel transfer on the Unibus. The control also provides the flags which cause a priority interrupt and indicate the availability of the teletype.

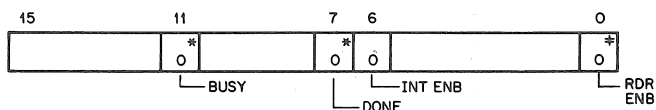
**KEYBOARD/READER—**The Teletype control contains an 8-bit buffer (TKB) which assembles and holds the code for the last character struck on the keyboard or read from the tape. Teletype characters from the keyboard/reader are received serially by the 8-bit shift register TKB. The code of a Teletype character is loaded into the TKB so that "spaces" correspond to binary 0's and holes, "marks," correspond to binary 1's. Upon program command, the contents of the TKB may be transferred in parallel to a memory location or a general register.

A character is read from the low-speed reader by setting the Teletype reader enable bit, (RDR ENB), to a 1. This sets the busy bit (BUSY) to a 1. When a Teletype character starts to enter, the control de-energizes a relay in the

Teletype unit to release the tape feed latch. When released, the latch mechanism stops tape motion only when a complete character has been sensed, and before sensing of the next character is started. When the character is available in buffer (TKB), the busy bit (BUSY) is cleared and the done flag (DONE) is set. If the interrupt is enabled, a request is made for the bus at level 4 (BR4). The interrupt vector is at location 60<sub>h</sub>. The DONE bit is cleared by any instruction which reads the contents of the buffer (TKB) into the processor. If the DONE flag is cleared before the interrupt is granted, no interrupt will occur. The keyboard must be read within 18 milliseconds of DONE to ensure no loss of information.

## Registers<sup>1</sup>

### Teletype Keyboard Status (TKS)



#### Bit

|    |         |  |
|----|---------|--|
| 0  | RDR ENB | Requests that one character be read from the reader; set from the bus: (Note: Setting RDR ENB causes tape to advance by one character which is shifted into TKB if DONE is cleared.) Receipt of START bit on the serial input line sets BUSY, clears RDR ENB and clears TKB. |
| 6  | INT ENB | 0—No interrupt; 1—Attach the keyboard and reader to the priority interrupt system at bus request level 4.  |
| 7  | DONE    | Character available; cleared by reading the buffer (TKB).  |
| 11 | BUSY    | Character is being read; set by RDR ENB going to a 1. Cleared by DONE going to a 1.  |

<sup>1</sup> The following notation will be used throughout this chapter for describing registers.

0 — A power clear sets this bit to 0.

1 — A power clear sets this bit to 1.

\* — This bit can only be read from the bus.

‡ — This bit can only be set from the bus. If it is read, it will always appear as zero.

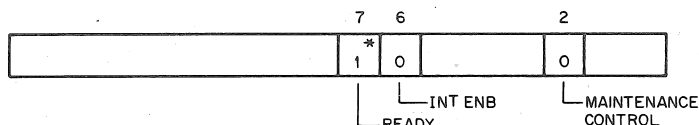
### Teletype Keyboard Buffer (TKB)



**TELEPRINTER/PUNCH**—On program command, a character is sent in parallel from a memory location (or a general register) to the TPB shift register for transmission to the teleprinter/punch unit. The control generates the start "space," then shifts the eight bits serially into the Teletype unit, and then generates the stop "marks." This transfer of information from the TPB into the teleprinter/punch unit is accomplished at the normal Teletype rate and requires 100 milliseconds for completion. The READY flag in the teleprinter/punch indicates that the TPB is ready to receive a new character. A maintenance mode is provided which connects the TPB output to the TKB input so that the parallel serial and serial parallel shifting may be verified.

## Registers

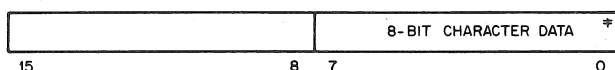
### Teleprinter Status Word (TPS)



#### Bit

|   |         |   |
|---|---------|---|
| 2 | MAINT   | Maintenance function which connects TPB serial output to TKB serial input.          |
| 6 | INT ENB | 0—No interrupt; 1—attaches the Teleprinter to the priority interrupt system at BR4. |
| 7 | READY   | Set by punch/printer DONE; cleared by loading the teleprinter buffer (TPB).         |

### Teleprinter Buffer (TPB)



**PROGRAMMING EXAMPLE**—To read a character from tape and echo it on the printer:

```
ECHO:      INC TKS                ; set RDR ENB
           TSTB TKS               ; test for DONE set
           BPL .—2                ; test again if not set
           TSTB TPS               ; test for printer READY set
           BPL .—2                ; test again if not set
           MOVB TKB, TPB          ; put input character into output
                                   buffer to be printed
           BR ECHO                ; return for another character
```

### PERIPHERAL ADDRESS ASSIGNMENTS

|     |        |
|-----|--------|
| TKS | 177560 |
| TKB | 177562 |
| TPS | 177564 |
| TPB | 177566 |

#### VECTOR ADDRESS

|                   |    |
|-------------------|----|
| Keyboard/Reader   | 60 |
| Teleprinter/Punch | 64 |

**PRIORITY LEVEL** set to BR4—Teletype printer is lower than the Teletype keyboard

**MOUNTING**—Requires one small peripheral controller mounting space ( $\frac{1}{4}$  of a DD11 or one of two such spaces in KA11)

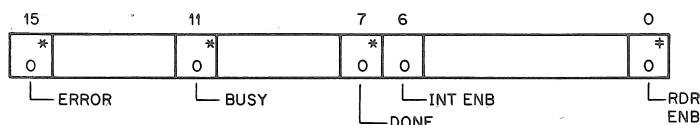
## HIGH-SPEED PERFORATED TAPE READER PUNCH AND CONTROL (TYPE PC11)

**TAPE READER**—This device senses 8-hole perforated paper or Mylar tape photo-electrically at 300 characters per second. The reader control requests reader movement, transfers data from the reader into the reader buffer (PRB), and signals the computer when incoming data is present. It does this

by setting a DONE bit. If the interrupt is enabled and the interrupt is granted, the processor traps to location 70, and may immediately begin executing the service routine for the paper tape reader.

## Registers

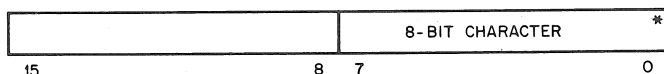
### Paper Tape Reader Status Word (PRS)



## Bit

|    |         |   |
|----|---------|---|
| 0  | RDR ENB | Requests read of next character; can be set from bus only if ERROR = 0. Clears PRB, sets BUSY.  |
| 6  | INT ENB | 0—No interrupt; 1—attached to priority interrupt system at BR4. (Note: Interrupt occurs when INT ENB is a 1 and either the error flag, ERROR, or the done flag, DONE, becomes a 1.) |
| 7  | DONE    | Set by character available; cleared by reading the paper tape reader buffer (PRB).  |
| 11 | BUSY    | Set by RDR ENB going to a 1; cleared by DONE going to a 1.  |
| 15 | ERROR   | Error Flag — Set or cleared by out-of-tape sensor or off line switch.   |

### Paper Tape Reader Buffer (PRB)



## PROGRAMMING EXAMPLE—Tape reading subroutine (not using interrupt):

```

READ:    INCB PRS          ; enable reader
TEST:    BIT #100 200 PRS  ; test for error or done
          BEQ TEST          ; branch back if not done
          BMI ERROR         ; branch if error = 1
          MOVB PRB, RO      ; get character from buffer
          RTS R             ; return to caller

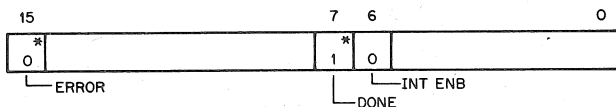
ERROR:    (message type out routine)
          HALT              ; wait for operator intervention
          JMP READ          ; try again when continue switch is hit.
```

**TAPE PUNCH**—This option of a Royal McBee paper tape punch that perforates 8-hole tape at a rate of 50 characters per second. Information to be punched on a line of tape is loaded in an 8-bit punch buffer (PPB) from a memory location or one of the general registers. The punch flag, READY, becomes a 1 at the completion of punching action, signaling new information may be transferred into the punch buffer and punching initiated.



## Registers

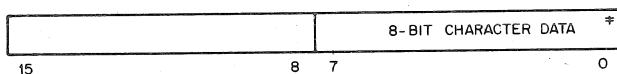
### Paper Tape Punch Status Word (PPS)



#### Bit

|    |         |   |
|----|---------|---|
| 6  | INT ENB | 0—No Interrupt; 1—Attached to priority interrupt system. (Note: An interrupt occurs when INT ENB is a 1 and either the ERROR flag or the READY flag becomes a 1.) |
| 7  | READY   | Set by punch done; cleared by loading the paper tape punch buffer (PPB).  |
| 15 | ERROR   | Error Flag—Set by out-of-tape sensor, or unit power off switch.   |

### Paper Tape Punch Buffer (PPB)



Loading the buffer initiates punching.

### PROGRAMMING EXAMPLE

```

PUNCH:   BIT #100200, PPS ; test for ready or error
          BEQ PUNCH
          BMI ERROR
          MOV R0, PPB      ;
          RTS R            ;

ERROR:    (message type out)
          HALT; wait for operator to fix punch
          JMP PUNCH; try again when Continue is hit.
    
```

### PERIPHERAL ADDRESS ASSIGNMENTS

```

PRS 177550
PRB 177552
PPS 177554
PPB 177556
    
```

**VECTOR ADDRESSES**—Reader 70  
Punch 74

**PRIORITY LEVEL**—Set to BR4. Punch is lower than reader.

**MOUNTING**—Electromechanical assembly—EIA Standard 19" rack, 10½" vertical mounting space, by 17½" deep.

PC11-M Controller—One small peripheral controller mounting space (¼ of DD11 or one of two such places in KA11).

## ENVIRONMENTAL

55°—100°F

20% —95% RH (without condensation)

## MODEL

## DESCRIPTION

## POWER REQUIREMENTS

|       |                         |
|-------|-------------------------|
| PC11  | Reader, Punch & Control |
| PC11A | Reader, Punch & Control |
| PR11  | Reader & Control        |

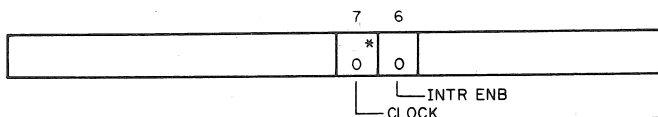
|         |          |
|---------|----------|
| 115±10% | 60 Hz    |
| 115±10% | 50 Hz    |
| 115±10% | 50-60 Hz |

## LINE FREQUENCY CLOCK (TYPE KW11-L)

The KW11-L real time clock provides a method of measuring time intervals at line frequency. This clock consists of a frequency source and control logic. When enabled the clock causes an interrupt every 16.6 or 20 milliseconds, depending upon line frequency.

## Register

Line Time Clock Status Register (LKS)



## Bit

**6 INTR ENB** When set, an interrupt will occur every time CLOCK goes true. Cleared by program or reset or start sequence.

**7 CLOCK** Set to 1 every 16.6 milliseconds (60 Hz) or 20 milliseconds (50 Hz). Cleared by reading LKS, RESET or pressing the START switch.

## PERIPHERAL ADDRESS ASSIGNMENTS

|                |        |
|----------------|--------|
| LKS            | 177546 |
| VECTOR ADDRESS | 100    |
| PRIORITY LEVEL | BR6    |

**MOUNTING**—This option plugs into the KA11 processor.

## CHAPTER 8

# DESCRIPTION OF THE UNIBUS

Communication between all system units in a PDP-11 configuration is done by a single common bus: the Unibus. All communication—both instructions and logical operations—is defined by a set of 56 signals. This set of 56 signals is used for program controlled data transfers, direct memory data transfers, priority bus control, and program interrupt.

This chapter presents the concepts of the Unibus and how they affect program software and interfacing hardware. The use of the 56 bus signals to effect data transfers and to control bus use is also described.

### GENERAL CONCEPTS OF THE UNIBUS

There are five major aspects of the Unibus that affect both software and hardware considerations in the PDP-11.

**SINGLE BUS**—The set of 56 signals that comprise the Unibus is the one and only bus connecting all peripheral devices, memories, and the central processor. Thus, to every device there exists a single set of signals by which it can be interrogated by the processor or other devices, or be used by the device itself to transfer data to and from memory.

The processor uses this same set of signals to communicate with all memories and devices. The important point here is that the form of the communication used by processor and peripheral devices is identical. Consequently, the same set of program instructions used to reference memory is used to reference peripheral devices. (A look at the PDP-11 instruction set will reveal that there are no explicit I/O instructions.)

Peripheral devices in a PDP-11 system are designed to respond to the Unibus in the same manner as memory. Device status registers, device control registers, and device data registers are each assigned unique “memory” addresses. For example, the instruction `MOVB R0, PUNCH` would load the punch buffer register with an 8-bit character contained in R0. Other instructions would monitor the punch status and the program could determine when the punching operation was complete.

**BIDIRECTIONAL BUS**—Unibus bus signals are bidirectional—the signal received as an input can be driven as an output, as shown in Figure 8-1.

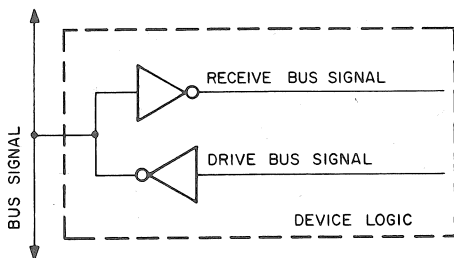


Figure 8-1 Bidirectional Nature of the Bus

**MASTER-SLAVE RELATION**—At any one point in time, there is one device, called the master, that has control of the bus. The master device controls

the bus to communicate with other devices, called slaves, on the bus. An example of this relationship is the processor (master) fetching an instruction from memory (which is always a slave).

**INTERLOCKED COMMUNICATION**—For each control signal issued by the master device, there is a response from the slave; thus bus communication is independent of the physical bus length and the response time of the master and slave devices. Also, master-slave relationships can exist in nearly any combination between fast-responding and slow-responding devices.

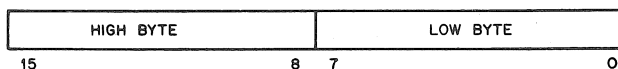
**DYNAMIC MASTER-SLAVE RELATION**—Master-slave relationships are dynamic. The processor, for example, can pass bus control to a disk. The disk, as master, could then communicate with a slave memory bank.

## UNIBUS SIGNALS

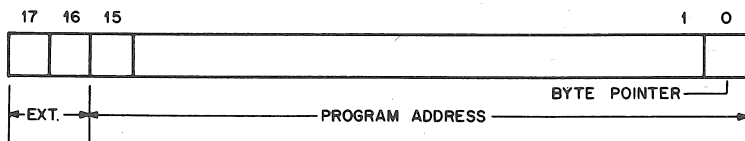
The 56 Unibus signals can be divided into two major groups—the interrupt group and the non-interrupt group. The interrupt group can then be subdivided into two classes—the request and control class and the grant class. All bus signals except the grant class are bidirectional in nature and are connected to every device (though they may not be used by every device). The grant signals, because of their special nature in priority bus control (to be explained later), are bussed through each device and are unidirectional in nature.

### NON-INTERRUPT SIGNALS

**Data Lines ( $D < 15:00 >$ )**—(Note that the notation  $A < a:b >$  specifies  $b - a + 1$  signal lines which are named  $Aa$  through  $Ab$ .) The 16 data lines are used to transfer information between master and slave. This is the bit format:



**Address Lines ( $A < 17:00 >$ )**—The 18 address lines are used by the master device to select the slave (a unique core memory or device register address) with which it will be communicating. This is the bit format of the 18 signals:



$A < 15:01 >$  are used to specify a unique 16-bit word group. In byte operations,  $A00$  is used to specify the byte being referenced. If a word is referenced at  $X$  ( $X$  must be even, since words can be addressed on even boundaries only), the low byte can be referenced at  $X$  and the high byte at  $X + 1$ .

$A < 15:00 >$  are supplied by the software as memory reference addresses.  $A17$  and  $A16$  are used as extended memory bits for relocation and as protection schemes in future systems. In the PDP-11/20 and the PDP-11/10,  $A17$  and  $A16$  are asserted or forced to 1 whenever an attempt is made to reference a memory location where  $A15 = A14 = A13 = 1$ . Thus the hardware converts the 16-bit software address to a full 18-bit bus address.

An address map is shown in Figure 8-2.

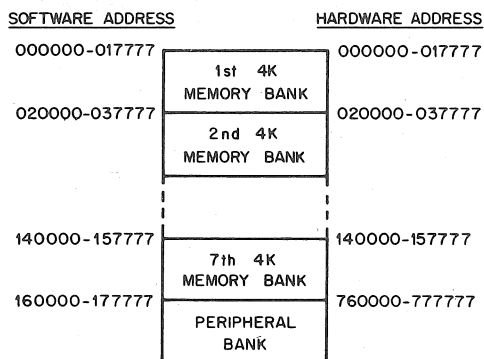


Figure 8-2 Address Map

The peripheral bank is composed of the processor's fast memory, status register, console switch register, and all device registers.

**Control Lines (C < 1:0 > )**—These two bus signals are coded by the master device to indicate to the slave one of four possible data transfer operations.

**Master Synchronization and Slave Synchronization (MSYN, SSYN)**—MYSN is a control signal used by the master to indicate to the slave that address and control information is present. SSYN is the slave's response to MSYN

**Initialization (INIT)**—This signal is a power clear signal asserted by the console and the processor which is used to reset peripheral devices.

**PA, PB, SP1, SP2**—These lines are not implemented on the PDP-11/10 or PDP-11/20.

## INTERRUPT SIGNALS

**Bus Request Lines (BR < 7:4 > )**—These four bus signals are used by peripheral devices to request control of the bus.

**Bus Grant Lines (BG < 7:4 > )**—These signals are the processor's response to a BR. They will be asserted only at the end of instruction execution.

**Non-Processor Request (NPR)**—This is a bus request from a peripheral device to the processor.

**Non-Processor Grant (NPG)**—This is the processor's response to an NPR. It occurs at the end of bus cycles within the instruction execution.

**Selection Acknowledge (SACK)**—SACK is asserted by a bus-requesting device that has received a bus grant. Bus control will pass to this device when the current master of the bus completes its operations.

**INTERRUPT (INTR)**—This signal is asserted by the master to start program interruption in the processor.

**Bus Busy (BBSY)**—This signal denotes bus in use by a master device.

## UNIBUS DATA TRANSFER OPERATIONS

Direction of data transfers on the Unibus is defined in relation to the master

device. A data transfer from processor to memory (always a slave) is "data out," and a transfer from memory to processor is "data in."

**TYPES OF DATA TRANSFERS**—The type of data transfer being made between master and slave is determined by the C lines coded as follows:

| C1 | CO |                        |
|----|----|------------------------|
| 0  | 0  | DATI - DATa In         |
| 0  | 1  | DATIP - DATa In, Pause |
| 1  | 0  | DATO - DATa Out        |
| 1  | 1  | DATOB - DATa Out, Byte |

**DATO AND DATOB**—The DATO and DATOB operations are used to transfer data out of the master to the slave. DATO is used to transfer a word to the address specified by A < 17:01 >. The slave ignores A00 and the data appears on D < 15:00 >. DATOB is used to transfer a byte of data to the address specified by A < 17:00 >. A00 = 0 indicates the low byte, and data appears on D < 07:00 >; A00 = 1 indicates the high byte, and data appears on D < 15:08 >.

**DATI AND DATIP**—The DATI and DATIP operations transfer data from a slave whose address is specified on A < 17:01 > into the master. Both transfers are made in words on D < 15:00 >. In destructive read-out devices, DATI commands a read-write operation, while a DATIP commands a read operation only and sets a pause flag. When the device receives the subsequent DATO or DATOB and its pause flag is set, the usual read cycle is skipped and an immediate write cycle is initiated. Thus, DATIPs are immediately followed by a DATO or DATOB to effect a read-modify-write data exchange. In non-destructive read-out devices, DATI and DATIP are treated identically.

This diagram illustrates the data flow in the four data transfers:

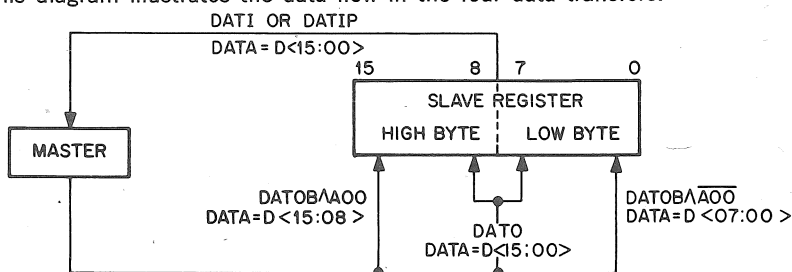


Figure 8-3 Data Flow

Note that all transfers into the master are word operations; it is up to the master to accept the appropriate byte. On a DATOB, the master must place the byte on the appropriate data lines; the slave must accept the proper byte.

**DATA TRANSFER EXAMPLES**—The bus operations used by the processor for a typical instruction sequence illustrates how the data transfer operations are used. The "program" starts at location 1000:

```
1000: INCB @R0
      ADD #3, @R0
```

where R0 contains 500 and location 500 contains 10023. The result of this

instruction sequence will leave 10027 in location 500. In binary form, this coding appears as:

```

1000:      105210      ;INCB @R0
1002:      062710      ;ADD (PC)+, @R0
1004:      000003      ;3

```

The following table lists the bus operations that result as a consequence of these two instructions:

| Processor Cycle | Bus Operation | Bus Address   | Data Transferred |
|-----------------|---------------|---------------|------------------|
| 1. Fetch        | DATI          | (PC) = 001000 | 105210           |
| 2. Destination  | DATIP         | (RO) = 000500 | 010023           |
| 3. Execute      | DATOB         | (RO) = 000500 | 000024           |
| 4. Fetch        | DATI          | (PC) = 001002 | 062710           |
| 5. Source       | DATI          | (PC) = 001004 | 000003           |
| 6. Destination  | DATIP         | (RO) = 000500 | 010024           |
| 7. Execute      | DATO          | (RO) = 000500 | 010027           |

Note that in step 3, it is inconsequential what data appears on D < 15:08 >; the slave accepts only the modified low byte.

A second example of bus operation compares the contents of the Teletype keyboard data buffer whose address is 177560 with the ASCII value for the letter "A."

```

200:      CMPB @#177560, #301

```

This instruction is assembled in three words as follows:

```

200:      123727      ;CMPB @(R7)+, (R7)+
202:      177560      ;Address of data buffer
204:      000301      ;301

```

The processor will execute this instruction with these cycles:

| Processor Cycle | Bus Operation                          | Bus Address | Data Transferred |
|-----------------|--|-------------|------------------|
| 1. Fetch        | DATI                                   | (PC) = 200  | 123727           |
| 2. Source       | DATI                                   | (PC) = 202  | 177560           |
| 3. Source       | DATI                                   | 777560      | ASCII            |
| 4. Destination  | DATI                                   | (PC) = 204  | 000301           |
| 5. Execute      | none — condition codes set internally. |             |                  |

Note that in step 3, the software specified address 177560 was converted to the bus address 777560.

**SIGNAL DESCRIPTION OF DATA TRANSFERS**—Figure 8.4(a) shows the signal flow between master and slave during a DATO operation. (The sequence is similar for DATOB except that only a byte of information is transferred.) The master sets Control for DATO, sets Address for the unique slave address, and sets Data for the information to be transferred. The master then asserts MSYN. This signal is received by the slave that recognizes its address; it responds by accepting the data and asserting SSYN. SSYN is received by the master which then negates Control, Address, Data, and MSYN. The slave sees MSYN negated and negates SSYN. The master device continues its operation when it sees SSYN negated.

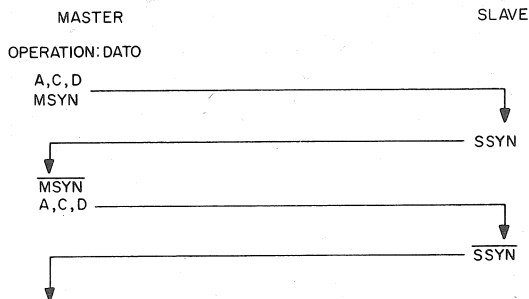


Figure 8-4(a)

The flow of signals for DATI is shown in Figure 8.4(b). (DATIP is similar except that the internal operation of the slave device is modified.) The master sets Control for DATI, sets Address for the slave to be selected, and asserts MSYN. The selected slave responds by setting Data for the information requested and asserts SSYN. The master sees SSYN, accepts the data, and then negates Control, Address, and MSYN. The slave sees MSYN negated and negates SSYN. The master continues when it sees SSYN negated.

A more detailed signal sequence for the DATI, DATIP, DATO, and DATOB bus operations can be found in Appendix D.

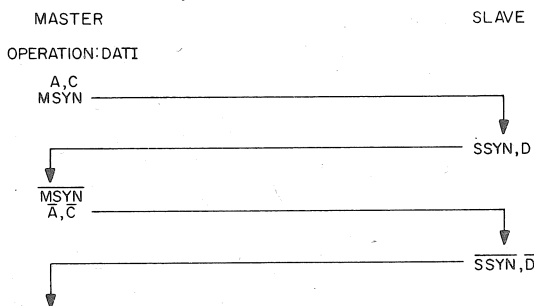


Figure 8-4(b)

## UNIBUS CONTROL OPERATIONS

The following section will deal with how a device becomes master of the bus and how control of the bus is transferred from one device to another. Two additional bus operations will be presented—the PTR (Priority Transfer) and INTR (Interrupt).

In normal operation, the processor is bus master, fetching instructions and operands from memory. Other devices on the bus have the capability of becoming bus master, and use the bus for one of two purposes: 1), to gain direct memory access or 2), to interrupt program execution and force the processor to branch to a specific address.

**PRIORITY ARBITRATION**—Transfer of bus control from one device to another is determined by a priority scheme in which three factors must be considered.

First, the processor's priority is determined by bits 7, 6, and 5 in the pro-



cessor status register. These three bits set a priority level that inhibits granting of bus requests on lower levels.

Second, bus requests from external devices can be made on one of five request lines. NPR has the highest priority, and its request is honored by the processor between bus cycles of an instruction execution. BR7 is the next highest; BR4 is the lowest. These four lower level requests are honored by the processor between instructions, except when the instruction currently being executed causes an internal trap (either an error or trap instruction). In this case, BR requests will not be honored until completion of the first instruction after the trap sequence. Thus if two requests are made to the processor for bus control, the higher of the two requests will be honored first.

Third, in response to a bus request, the processor may honor the request by asserting a bus grant (BG) corresponding to the line on which the bus request was made. This signal is passed serially through each device in the system. If a device had made a request, it would block the grant signal and prevent it from reaching the following devices. Thus, in this "pass-the-pulse" chain, the device that is closest to the processor has the highest priority on that request level.

This table lists device priorities:

|          |                               |
|----------|-------------------------------|
| Highest: | Devices on NPR                |
|          | Processor when priority = 111 |
|          | Devices on BR7                |
|          | Processor when priority = 110 |
|          | Devices on BR6                |
|          | Processor when priority = 101 |
|          | Devices on BR5                |
|          | Processor when priority = 100 |
|          | Devices on BR4                |
|          | Processor when priority = 011 |
|          | Internal options              |
|          | Processor when priority = 010 |
|          | Internal options              |
|          | Processor when priority = 001 |
|          | Internal options              |
| Lowest:  | Processor when priority = 000 |

When the processor's priority is set at N, all requests for bus control at level N and below are ignored.

**SELECTION OF NEXT BUS MASTER**—The signal sequence by which a device becomes selected as next bus master is the PTR (Priority Transfer) bus operation. Note that this operation does not actually transfer bus control; it only selects a device as next bus master. It takes one additional condition to complete the transfer: the current bus master must complete its bus operations. The signal that indicates this is BBSY. Thus, when a device makes an NPR or BR request to the processor for bus control, it waits until it first becomes selected as next bus master by the PTR operation and second, it no longer senses BBSY. The negation of the BBSY signal indicates that the current master has completed its bus operation. The selected device now becomes bus master and asserts BBSY itself.

**INTERRUPT SEQUENCE**—Once the device has bus control and is asserting BBSY itself, it is sole user of the bus until it releases its control. This release of control can be made either actively or passively. Passive release is realized

by negating BBSY. Bus control will then pass to either a device that was selected in the meantime by another PTR sequence or back to the processor, which will continue where it was interrupted. Active release of bus control is realized through the INTR bus sequence.

The INTR (interrupt) operation is used by the bus master to transfer to the processor a memory address (called the interrupt vector). Two consecutive words, the starting address of an interrupt service routine and a new status word, are stored at the interrupt vector address. After the INTR sequence is complete, the processor automatically becomes bus master and begins a trap sequence in which it stores the current value of the PC and PS on the stack and fetches a new PC and PS from the location pointed to by the interrupt vector. Thus, the next instruction executed is the start of the interrupt service routine.

It is illegal to issue an INTR command after gaining control of the bus by requesting on an NPR line. NPR requests are granted during instruction execution and external bus masters must restrict their bus use to non-processor activities.

**Interrupt Servicing Sequence Example**—The following is an example of the INTR sequence.

When a peripheral requires service and requests control of the bus with a BR signal, the operations undertaken to “service” the device are as follows:

- **Gain Control of the Bus**—When the processor has no higher priority tasks to complete, it relinquishes the bus to that device. Higher priority items are (in order of priority):

1. Acknowledging an NPR request
2. Handling a processor error (illegal instructions, requirements for non-existent memory, etc.)
3. Completing the current instruction
4. Acknowledging a trace trap
5. Continuing a higher priority process
6. Acknowledging a higher level BR signal
7. Acknowledging same level BR signals for devices closer to the processor

- **Do INTR Sequence**—when the device has control of the bus, it initiates an INTR sequence, transferring to the processor the interrupt vector address which specifies two words in memory containing the address and status of the appropriate device service routine.

- **Push Old Interrupt Vector Onto Stack**—The processor then “pushes”—first, the current central processor status (PS) and then the current program counter (PC) onto the processor stack.

- **Fetch New Interrupt Vector**—The new PC and PS (the “interrupt vector”) are taken from the address specified by the device, and the device service routine is begun. Note that those operations all occur automatically and that no device polling is required to determine which service routine to execute.

**Example of NPR Operation**—Disk operation gives an example of a device which uses the bus for direct memory access. Under program control, the processor would initialize registers in the disk control that specify word count (WC, number of words in block of data to be transferred), memory address (MA, the address at which the block of data is found or is loaded), and Track Address (TA, the point on the disk where the block of data starts). Also, the

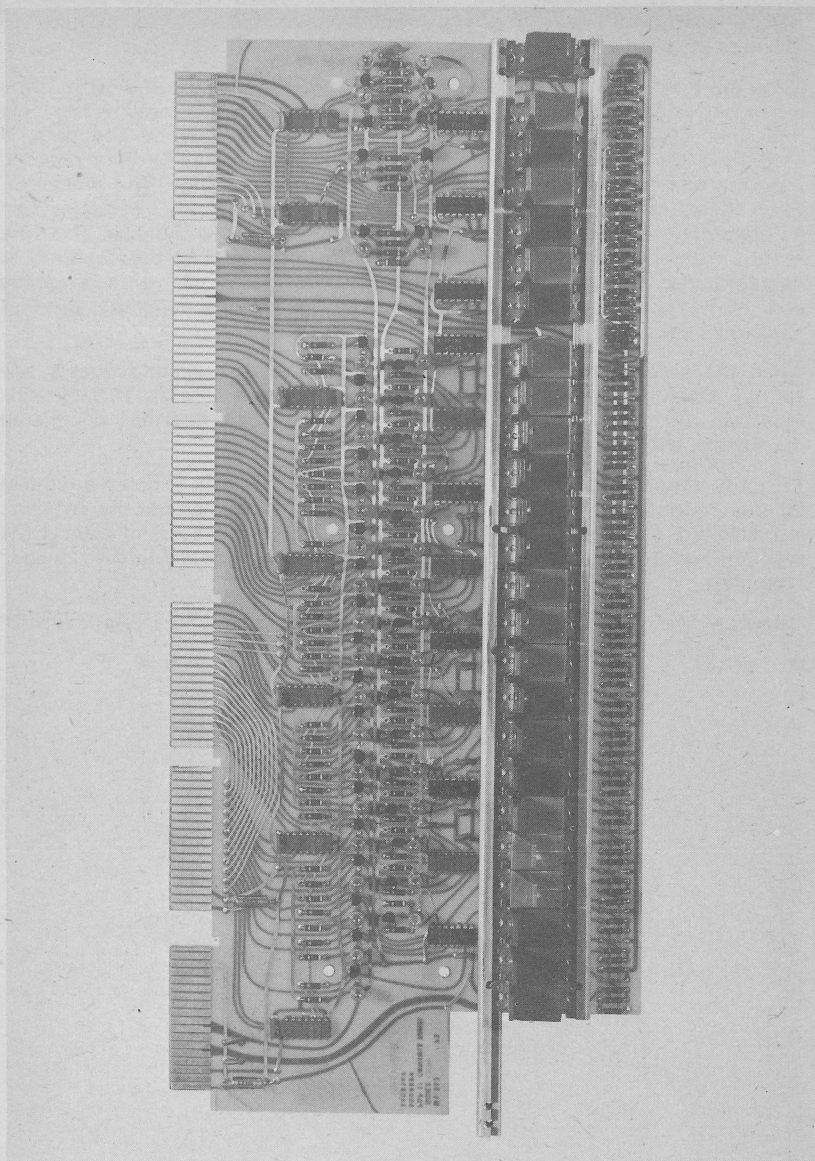
program would set certain function bits in the disk's command and status register that specify a read or write function. For this example, assume the disk was set to read.

Once the disk's control registers are initialized, the disk control logic starts a search for the requested data. (The processor in the meantime has continued in its program execution.) When the disk has found the data, it assembles the first 16-bit word from the disk surface into its data register. The disk now requests bus control via the NPR request line. The processor, when it has completed its current bus cycle of the current instruction and no higher NPR requests exist, grants control of the bus to the disk. The disk, as bus master, effects a DATO bus operation, transferring the contents to its data buffer to the core address held in its MA. The MA is now incremented and the WC is decremented. When the DATO operation is complete, the disk passively releases control of the bus.

When the second word has been assembled, the disk again requests bus control, does a data transfer, and then releases bus control. This cycle is repeated until the WC reaches zero. At this point, the disk has completed the transfer that was requested.

To notify the program that the transfer is finished, the disk initiates a request for bus control at the BR level, gains control when higher priority requests are satisfied, and does an immediate INTR to the processor and causes the program to branch to a specific service program (as described in the previous example).

Details of the INTR and PTR bus operations can be found in Appendix D.



The plug-in console board with modular construction is supplied in the basic 11/20 configuration. In addition to aiding programming, console contributes to ease of maintenance on the PDP-11.

## CHAPTER 9

### Interfacing

A typical device bus interface as shown in Figure 9-1 is composed of five major components: 1), Registers; 2), Bus Drivers and Receivers; 3), Address Selector; 4), Interrupt Control; and 5), Device Control Logic.

#### REGISTERS

Each device is assigned bus addresses at which the program can interrogate and/or load the device status, control, and data registers. The standardized mapping for these registers and the bit assignments of the command/status register (CSR) were given in Chapters 5 and 6.

As shown in Figure 9-1, all information flow between the device logic and the Unibus is done through the registers. In general, registers are designed to be both loadable and readable from the bus. This allows the program to use such instructions as ADD R0, REG, or INC REG. However, registers can be "one-sided," either "read-only" or "write-only." Examples of read-only bits are the DONE and BUSY flags in the device's CSR. These bits are derived from the internal state of the device logic and are not under direct program control. Write-only registers are used when it is unnecessary to read back information. Attempting to read such a register would result in an all-zero transfer. The instructions effective with this type of register are then limited to those which load the register such as MOV R0, REG, or CLR REG (as opposed to ADD REG, R0, or INC REG).

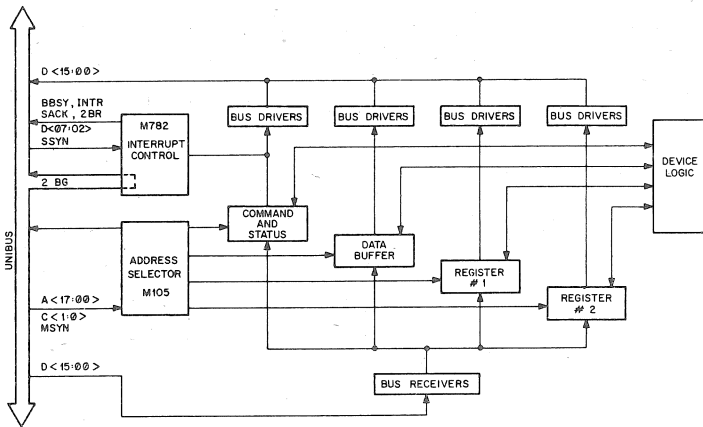
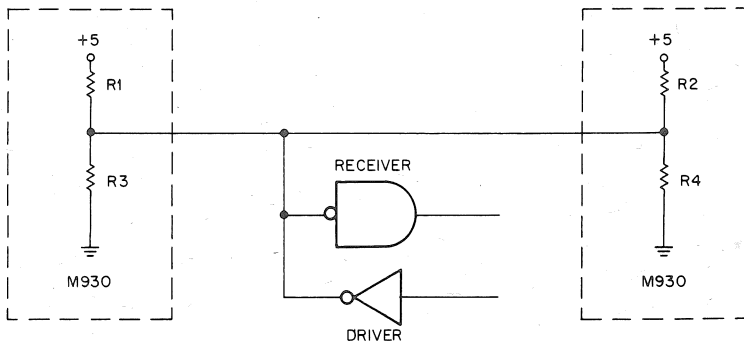


Figure 9.1 Typical Peripheral Device Interface

#### BUS DRIVERS AND RECEIVERS

To maintain the transmission-line characteristics of the Unibus, special circuits are required to pass signals to and from the bus. The majority of bus signals (all except the five grant lines) are received, driven and terminated as shown in Figure 9-2.



R1, R2 = 180  $\Omega$  5% 1/4W

R3, R4 = 390  $\Omega$  5% 1/4W

Figure 9.2 Typical Unibus Line

Information is received from the bus using gates which have a high input impedance and proper logic thresholds. High input levels must be greater than 2.5 V with an input current less than 160  $\mu$ a. Low level input must be less than 1.4 V with an input current greater than 0  $\mu$ a.

Information transmitted on the bus must be driven with open collector drivers capable of sinking 50 ma with a collector voltage of less than .8 V. Output leakage current must be less than 25  $\mu$ a.

In PDP-11 systems, the bus signals are terminated at both ends by resistor dividers provided on the M930 module. Physically, an M930 is located in the processor; another is located at the last unit on the bus. A bus signal sits at logical "0" (inactive, or negated state) at a voltage of 3.4 V. A bus line is at logical "1" (active, or asserted) when it is pulled to ground.

Drivers and receivers meeting these specifications are available on the M783, M784 and M785 modules as shown in Figures 9-3, 9-4 and 9-5.

## M105 ADDRESS SELECTOR

The M105 Address Selector as shown in Figure 9-6 is used to provide gating signals for up to four device registers. The selector decodes the 18-bit bus address on  $A < 17:00 >$  as follows:

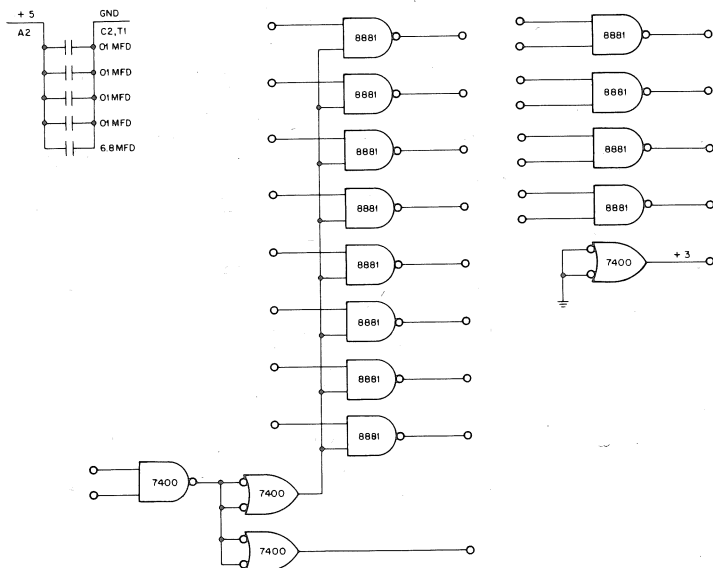


Figure 9.3 M783 Unibus Drivers

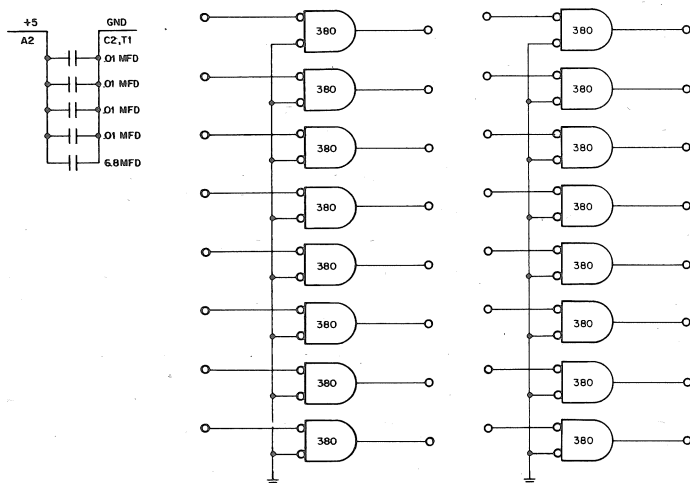


Figure 9.4 M784 Unibus Receivers

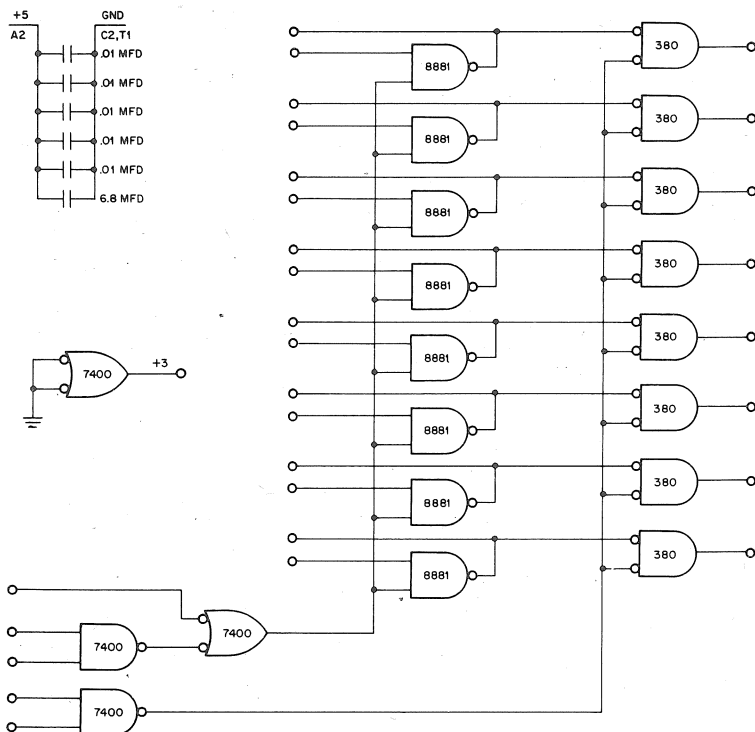


Figure 9.5 M785 Unibus Drivers and Receivers

A00 is used for byte control. A01 and A02 are decoded to provide one of four addresses. A < 12:03 > are determined by jumpers on the card. When the jumper is in, the selector will look for a 0 on that address line. A < 17:13 > must all be 1's—(this defines the external bank). Other bus inputs to the selector are C < 1:0 > and MSYN. The single bus output is SSYN. The user signals are SELECT 0, 2, 4, and 6 (corresponding to the decoding of A02 and A01, one of which is asserted when A < 17:13 > are all 1's and A < 12:03 > compare with the state of the jumpers. Other user signals are OUT HIGH (gate data into high byte), OUT LOW (gate data into low byte), and IN (gate data onto the bus). The equations for these last three signals are as follows:

$$\begin{aligned} \text{OUT HIGH} &= \text{DATO} + \text{DATOB} * \text{A00} \\ \text{OUT LOW} &= \text{DATO} + \text{DATOB} * \text{A00} \\ \text{IN} &= \text{DATI} + \text{DATIP} \end{aligned}$$

where "+" means a logical or and "\*" means a logical and.

Use of the M105, drivers, receivers and a flip-flop register is shown in Figure 9-7.



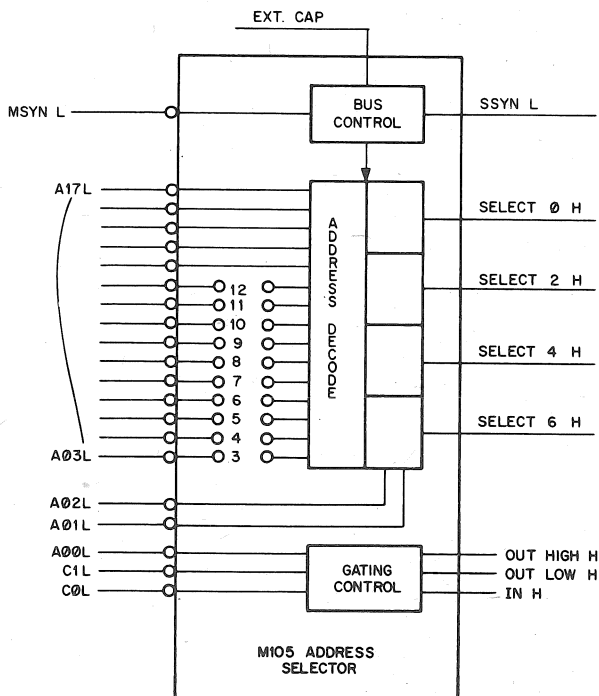


Figure 9.6 M105 Address Selector

## M782 INTERRUPT CONTROL

The M782 Interrupt Control module contains the necessary logic circuits to allow a peripheral device to gain bus control and perform a program interrupt. The three circuits on this card are block diagrammed in Figure 9-8. Note that only signals relevant to the user's interface are shown; bus signals SSYN, BBSY and SACK have been omitted for clarity.

The Master Control circuit is used to gain bus control. When INT and INT ENB are asserted, a bus request is made on the request line to which BR is jumpered. When the processor issues the corresponding grant and other bus conditions are met, the MASTER signal is asserted, indicating that this device now has bus control. Note that this circuit also can be used to gain bus control on an NPR line for a device which requests the bus for direct memory access.

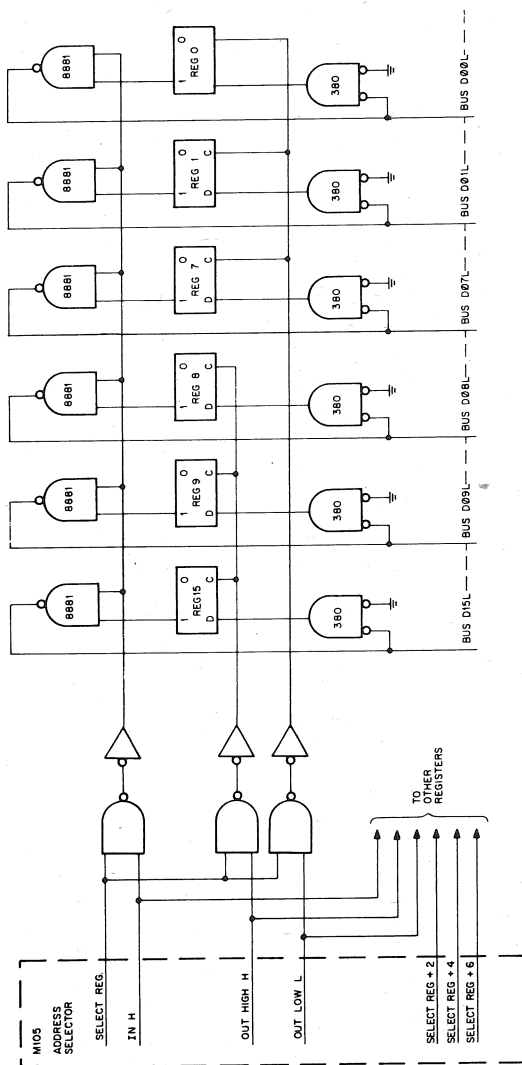


Figure 9.7 Typical Peripheral Device Register

In addition to two Master Control circuits, a third logic network provides the necessary signals and gating to perform the INTR bus operation. When either of the START INTR signals is asserted, the INTR bus signal is asserted along with a vector address on  $D < 07:02 >$ . Bits 07:03 are determined by jumpers on the card. A jumper "in" forces a 0 in that bit. Bit 2 is controlled by Vector Bit 2. When the processor responds to the INTR signal by asserting SSYN, the INTR DONE signal is asserted. This line is used to clear the condition which asserted INTR START.

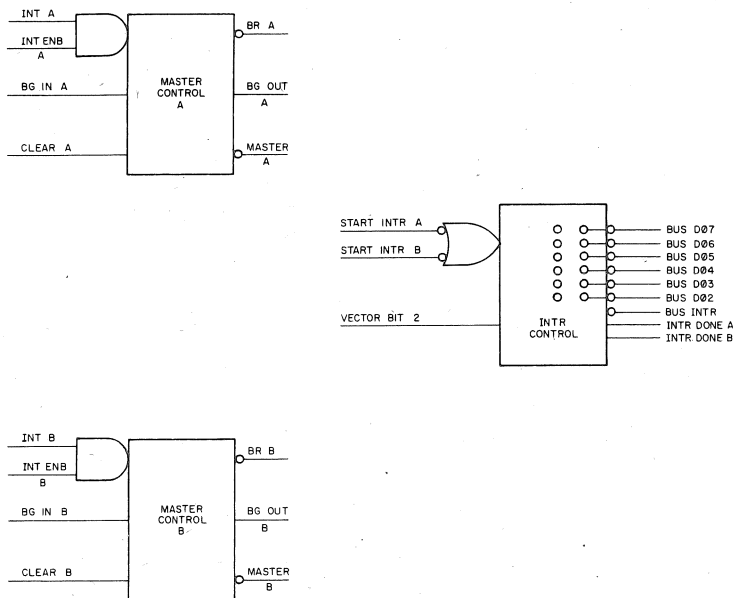


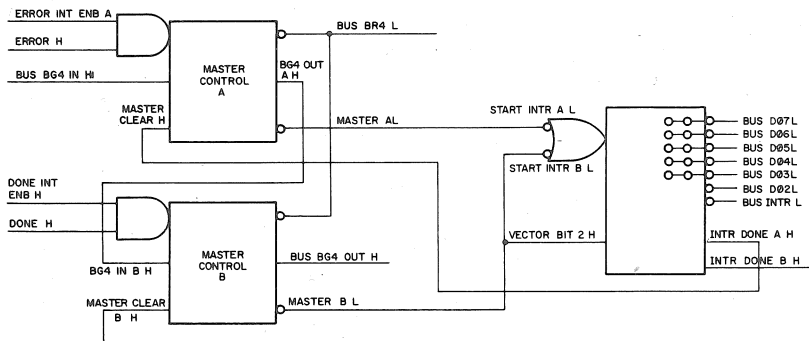
Figure 9.8 M782 Interrupt Control

Figure 9-9 shows a possible interconnection of the M782 to provide independent interrupts for two possible conditions in a device: ERROR and DONE. The ERROR and DONE signals shown in Figure 9-9 are signals from bits 15 and 7 in a device's CSR. Likewise ERROR INT ENB and DONE INT ENB are derived from the CSR. Both interrupts in this example are tied to the BR4 level; the corresponding grant line BG4 enters the ERROR Master Control and is passed on to the DONE Master Control. Thus, ERROR has a slightly higher priority interrupt level than DONE.

Both MASTER signals are tied to the INTR control. Thus, whenever either ERROR or DONE gains bus control, an INTR operation is initiated. Note that Vector Bit 2 is a 1 or 0 as a function of which master control is interrupting. Also, INTR DONE is tied to MASTER CLEAR to clear the master condition.

## DEVICE CONTROL LOGIC

The type of control logic for a peripheral depends on the nature of the device. Digital offers a wide line of general-purpose logic modules for implementing control logic. These modules are described in detail in another Digital publication: The Logic Handbook.



- NOTE:
1. BUS REQUEST IS MADE ON LEVEL 4
  2. "ERROR" INTERRUPTS TO 104
  3. "DONE" INTERRUPTS TO 100
  4. "ERROR" HAS HIGHER PRIORITY THAN "DONE" BECAUSE "ERROR" RECEIVES BG4 FIRST.

Figure 9.9 Typical Interconnection of M782 Interrupt Control

# CHAPTER 10

## CONFIGURATION AND INSTALLATION PLANNING

### MODULAR CONSTRUCTION

Physically, the PDP-11 is composed of a number of System Units. Each System Unit is composed of three 8-slot connector blocks mounted end-to-end as shown in Figure 10-1. The Unibus connects to the System Unit at the lower left and at the upper left. Power also connects to the unit in the leftmost block. A System Unit is connected to other System Units only via the Unibus.

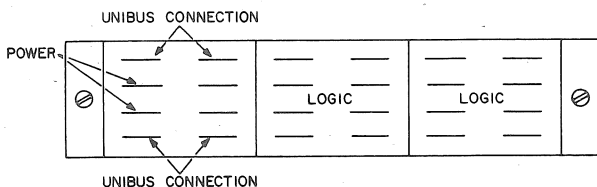


Figure 10.1 System Unit

The remainder of the System Unit contains logic for the processor, memory or an I/O device interface. This logic is composed of single height, double height, or quad height modules which are 8.5" deep.

The use of System Units allows the PDP-11 to be optimally packaged for each individual application. Up to six System Units can be mounted into a single mounting box. For a basic PDP-11/20 system, the processor/console would fill  $2\frac{1}{2}$  System Unit spaces and 4096 words of core memory would fill one System Unit space. This leaves  $2\frac{1}{2}$  spaces for user-designated options. This would allow the user to add 8,192 words of additional core memory, a Teletype control, and a High-Speed Paper Tape Control, or 4,096 words of core memory and six Teletype interfaces. Larger systems will require a BA11-EC or BA11-ES Extension Mounting Box which contains space for six additional System Units.

The use of System Units also facilitates expansion of systems in the field and service. To add an additional option to a PDP-11 system, the proper System Unit is mounted in the Basic or Extension Mounting Box and the Unibus is extended. Servicing of the PDP-11 can be done by swapping modules or by swapping System Units.

### MOUNTING BOXES AND CABINETS

The PDP-11 is available as either a tabletop or rack-mounted configuration. The rack-mounted configuration may be installed in a DEC cabinet or mounted in a customer cabinet. The PDP-11 mounts in an EIA standard 19-inch cabinet. The rack-mounted PDP-11 has tilt-slides as standard mounting hardware.

The following mounting units and cabinets are available for PDP-11 systems.

**PDP-11 TABLETOP BOX AND POWER SUPPLY FOR 11/20, 11/10 SYSTEMS (BA11-CC AND H720)**—This cover and box may be specified with a basic 11/20 and 11/10 system and includes:

1. H720 Power Supply
2. 15' of power cord with ground wire

- For 115 V standard, 3-prong, U-ground, 15-ampere connectors
- For 230 V pigtail leads on one end
- 3. Cooling Fans
- 4. Filter
- 5. Programmers Console with 11/20 or Turn-Key Console with 11/10

Approximate Size—11" high, 20" wide, 24" deep. Figure 10-2 shows the layout of this unit.

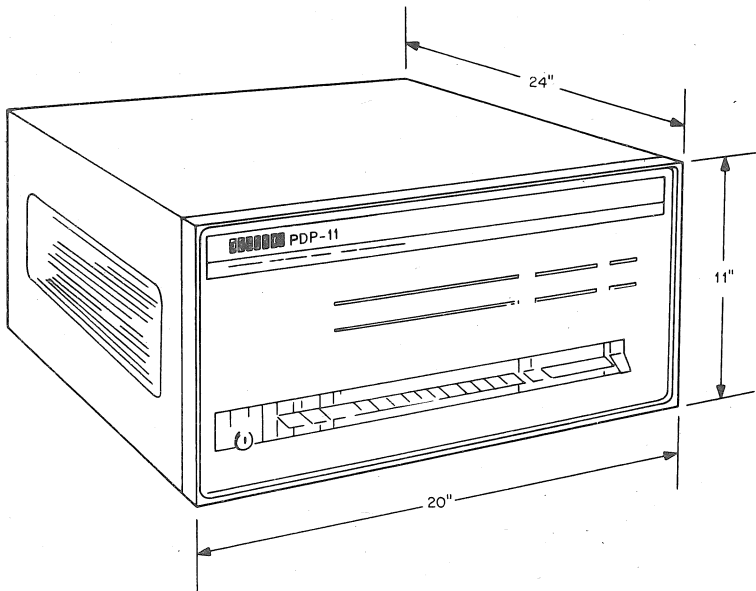


Figure 10.2 Table Top PDP-11 Dimensions

Approximate Weight—100 lbs. (including CP, console and 4K core)

|                                 |         |              |
|---------------------------------|---------|--------------|
| Power—120 V $\pm$ 10%, 47-63 Hz | 6 amps. | single phase |
| (BA11-CC and H720-A)            |         |              |
| 230 V $\pm$ 10%, 47-63 Hz       | 3 amps. | single phase |
| (BA11-CC and H720-B)            |         |              |

#### **PDP-11 BASIC MOUNTING BOX AND POWER SUPPLY (BA11-CS AND H720)**

—This basic mounting box may be specified with a basic 11/20 or a 11/10 system and includes:

1. Tilt and Lock Chassis Slides
2. H720 Power Supply
3. 15' of power cord with ground wire
  - For 115 V standard, 3-prong, U-ground, 15-ampere connector
  - For 230 V pigtail leads on one end
4. Cooling Fans
5. Filter
6. Programmers Console with 11/20 or Turn-Key Console with 11/10

Approximate Size—10½" high, 19" wide, 23" deep. Figures 10-3, 10-4 and 10-5 show the layout of this unit and give slide dimensions.

Approximate Weight—90 lbs. (including CP, console and 4K core)

|                                 |         |              |
|---------------------------------|---------|--------------|
| Power—120 V $\pm$ 10%, 47-63 Hz | 6 amps. | single phase |
| (BA11-C5 and H720-A)            |         |              |
| 230 V $\pm$ 10%, 47-63 Hz       | 3 amps. | single phase |
| (BA11-C5 and H720-B)            |         |              |

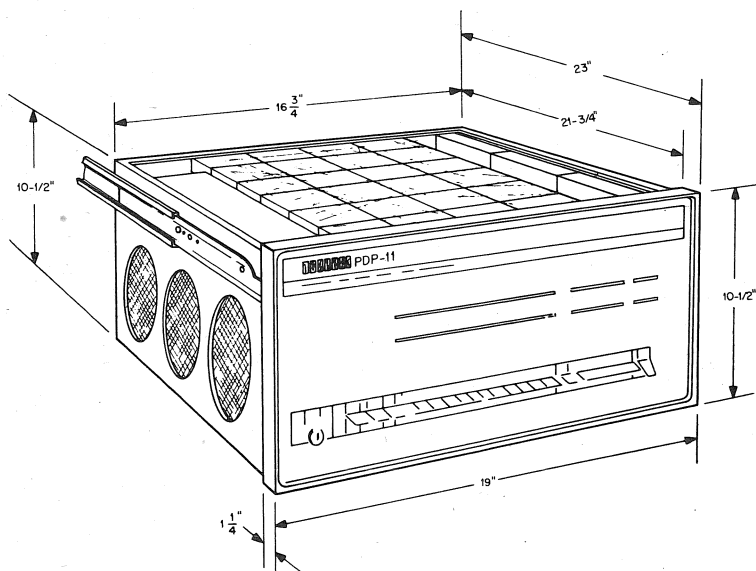


Figure 10.3 Rack Mountable PDP-11 Dimensions

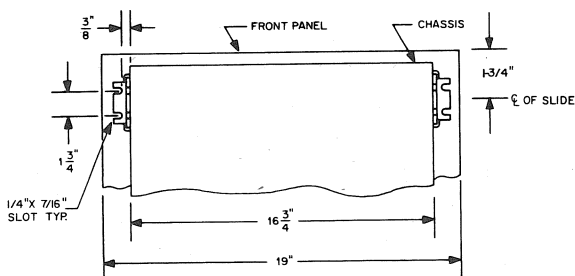


Figure 10.4 Rear View of Mounting Hardware

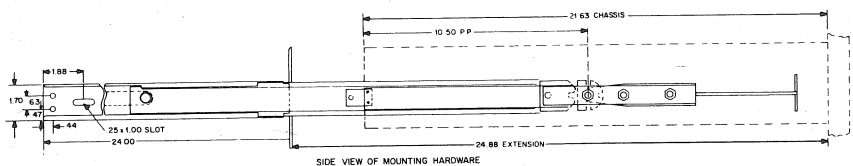


Figure 10.5 Side View of Mounting Hardware





10. H950-D Mounting Panel Doors
11. H952-B Stabilizer Feet
12. #7406782 Kick Plate
13. #7005909 Power Distribution Panel (ac and dc, mounted on upper left side)

Approximate Size—22" wide, 39" deep (including stabilizer feet), 71½" high

Approximate Weight—150 lbs. (without computer)

Voltage—115 V 60 Hz (for fans)

230 V 50 Hz (for fans)

**PDP-11 POWER SUPPLY SUBSYSTEM H720**—This Power supply is used in the Basic and Extension Mounting boxes and supplies power to all devices mounted in one of these boxes. It is included in basic PDP-11 systems, but must be ordered separately with a BA11ES or BA11EC Extension Mounting Box.

Approximate Size—16½" wide, 8" high, 6" deep

Approximate Weight—25 lbs.

Voltages—(specify input voltage)

|     |                     |          |         |
|-----|---------------------|----------|---------|
| IN  | 108V ±10%, 47-63 Hz | 6 amps   | (H720A) |
|     | 120V ±10%, 47-63 Hz | 6 amps   | (H720A) |
|     | 216V ±10%, 47-63 Hz | 3 amps   | (H720B) |
|     | 228V ±10%, 47-63 Hz | 3 amps   | (H720B) |
|     | 240V ±10%, 47-63 Hz | 3 amps   | (H720B) |
| OUT | +5V ±5%             | 12 amps  |         |
|     | −15V ±5%            | 10 amps  |         |
|     | +8RMS (unregulated) | 1.5 amps |         |
|     | −22V (unregulated)  | 1.0 amps |         |

**FREESTANDING PROGRAMMER'S TABLE (H952-HA)**—This freestanding table fits directly below the programmer's console in the Freestanding Base Cabinet and extends into the cabinet approximately 1". The surface plate is supported by its own adjustable height legs.

Approximate Size—20" extension from cabinet, 19" wide, 27" above floor

## SYSTEM UNITS AND CABLES

The following items are available for mounting standard and special peripheral device logic into a PDP-11 system.

**PERIPHERAL MOUNTING UNIT (DD11-A)**—The DD11 is a prewired System Unit which allows standard small peripheral interfaces to be mounted in a PDP-11 system. It accepts standard small peripheral interfaces (up to 4) such as the KL11 Teletype Control or the controller portion (PC11-M) of the High Speed Reader/Punch. For mounting, it requires one-sixth (1/6) of a BA11 Mounting Box.

**BLANK SYSTEM UNIT (BB11)**—The BB11 consists of three 288-pin connector blocks connected end-to-end. This unit is unwired except for Unibus and power connections and allows customer-built interfaces to be integrated easily into a PDP-11 system. For mounting it requires one-sixth (1/6) of a BA11 Mounting Box.

**UNIBUS MODULE (M920)**—The M920 is a double module which connects the Unibus from one System Unit to the next within a Mounting Box. The printed circuit cards are separated by 1" for this purpose. A single M920 will carry all 56 Unibus signals and 14 grounds.

**UNIBUS CABLE (BC11A)**—The BC11A is a 120-conductor flexprint cable used to connect System Units in different mounting boxes or a peripheral device which is removed from the mounting boxes.

The 120 signals consist of the 56 Unibus lines plus 64 grounds. Signals and grounds alternate to minimize cross talk.

| Type     | Length |
|----------|--------|
| BC11A-2  | 2'     |
| BC11A-5  | 5'     |
| BC11A-8A | 8'6"   |
| BC11A-10 | 10'    |
| BC11A-15 | 15'    |
| BC11A-25 | 25'    |

## **CABLE REQUIREMENTS**

When an Extension Mounting Box is used, an external cable, the BC11A, is the only signal connection between mounting boxes. This external bus cable may also be used to connect other peripherals to the PDP-11. The maximum combined, internal and external, bus cable length is 50'.

## **PDP-11/20 POWER REQUIREMENTS**

Input Voltage and Current—105-125 Vac, 6 amperes, 210-260 Vac 3 amperes, (single phase)

Line Frequency—47-63 Hz

Power Dissipation—400 watts

A standard 15-foot, 3-prong, U-ground, 15-ampere, line cord is provided on the rear of the PDP-11 for connection to the power source on 120 Vac models. On 230 Vac models, a 15-foot, 3-conductor cable with pigtails is provided.

## **TELETYPE REQUIREMENTS**

The standard Teletype requires a floor space approximately 22½ inches wide by 18½ inches deep. The Teletype cable length restricts its location to within 8 feet of the side of the computer.

Input Voltage—115 Vac  $\pm 10\%$ , 60 Hz  $\pm 0.45$  Hz, 230 Vac  $\pm 10\%$ , 50 Hz  $\pm 0.75$  Hz

Line Current Drain—2.0 amperes

Power Dissipation—150 watts

The Teletype plugs into the rear of the PDP-11 Basic Mounting Box and is turned ON and OFF by the power switch on the front panel of the PDP-11.

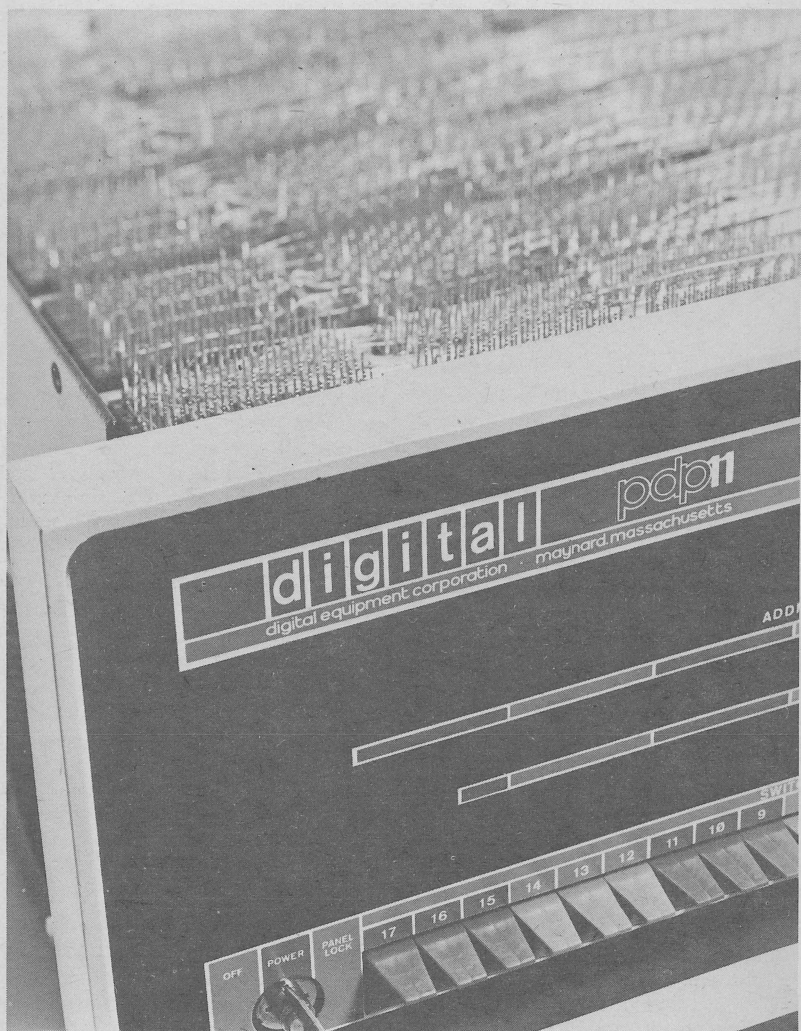
## **ENVIRONMENTAL REQUIREMENTS**

The PDP-11 is designed to operate from +10 to +50°C and with a relative humidity of from 20 to 95% (without condensation).

## **INSTALLATION PROCEDURE**

The PDP-11 is crated for shipment to the customer site to prevent damage. Installation is provided by DEC personnel at the customers site.

Computer customers may send personnel to instruction courses on computer operation, programming, and maintenance conducted regularly in Maynard, Massachusetts, Palo Alto, California, and Reading, England.



The PDP-11 has adopted a modular packaging approach to allow custom configuring of systems, easy expansion and easy servicing.

# CHAPTER 11

## PAPER TAPE SOFTWARE SYSTEM

### PAPER TAPE SOFTWARE SYSTEM (PTS)

PTS is a compatible group of software packages designed to aid development of PDP-11 application programs. A brief description of each item with its major features is offered below with detailed programming information available in corresponding software user's manuals.

#### PTS FEATURES

- 4K Absolute Assembler
- Symbolic Program Editor for editing of paper tape which is string oriented
- On-Line Debugging Aid allowing rapid and accurate modification of assembled programs
- I/O Driver Routine allowing subroutine level communication with peripheral devices and double buffered input/output operation concurrent with running programs
- Floating Point Math Package using both reentrant and relocatable code
- General Utilities including loaders and dump routines

**PAL-11A ASSEMBLER**—This two- or three-pass assembler runs on a PDP-11 with 4K words of core memory and an ASR-33. It will also accommodate a high-speed reader/punch. Optional outputs include the absolute object code, an assembly listing containing each source statement, and an indication of any errors detected in the statement. A symbol table may be alphabetically listed.

**ED11 EDITOR**—The PDP-11 Editor (ED11) allows the user to type identified portions of source program on the teleprinter and to make corrections or additions. This is accomplished by typing simple commands that cause the Editor to read, print, punch out on paper tape, search, delete and/or add to the text of the program.

Use of the ED11 presupposes no special knowledge or technical skill beyond that of the operation of explicitly defined one-character commands. The commands are grouped according to function: input, positioning of the current-character location pointer, output, search (which is done by character string), insert, delete, and exchange of text portions.

ED11 uses 2,000 words of core and requires an ASR-33 unit which includes a printer, keyboard, paper tape reader and paper tape punch. Alternatively, a KSR-33 may be used in conjunction with the high-speed paper tape reader and punch.

**ODT-11 ON-LINE DEBUGGING TECHNIQUE**—ODT-11 is a core resident program which allows the user to debug his binary programs at the console by running them in specific segments and checking for expected results at various points. If modification of the program is needed, the user can alter the contents of the appropriate location by "opening" it and typing in new data.

Two versions of ODT are available, one being a subset of the other. The larger system uses 750 words of core and utilizes an ASR-33, or a KSR-33 and a high-speed paper tape punch and reader. The smaller version uses the same peripherals and 500 words of core. Up to eight breakpoints can be set using the larger version of ODT, while one breakpoint is allowed in the smaller version.

Debugging operations alternate between commands to ODT and the running of the program to be debugged. Breakpoints are set in the user's program by ODT commands, and a command to run starts execution of the program. When a breakpoint is encountered, the program run is suspended, and the progress of its execution can be monitored and altered. This is accomplished by using commands to open memory locations of interest, as well as special registers.

An operator may examine and change the operating priority of both ODT and the user's program, the mask and address range for searches, results of logical and arithmetic operations, the SP and PC, and the general registers. Other commands will search for values of specified bits of a word, or for references to an address within an address range, calculate 16-bit and 8-bit offsets to an address and restart the running of the user's program at any address.

**IOX Input/Output Utility Peripheral Driver**—IOX is a set of service routines allowing single or double buffered I/O processing on an ASR-33 and/or a high-speed paper tape reader and punch. This routine allows the user to make simple assembly language calls specifying devices and data forms to accomplish interrupt-controlled data transfer concurrent with execution of the running program. Multiple devices can be run simultaneously.

IOX frees the user from the details of dealing directly with the device and allows development of programs which may be run under the direction of a monitor with minimum modification.

IOX also provides some degree of real-time control by allowing user programs to be executed at priority levels at the completion of some device action or data transfer.

**MATH PACKAGE**—A number of commonly used subroutines are available to simplify programming. These routines are reentrant and relocatable to provide maximum flexibility. Arguments are treated as floating point numbers with a signed 31-bit fraction and a signed 15-bit exponent. Subroutines supplied include:

- ADD
- MULTiply
- SUBtract
- DIVide
- SIN
- COS
- ATAN
- FIX—FLOAT
- FLOAT—FIX
- NORMALize
- (Integer MULTiply and DIVide are also supplied)

**LOADERS**—Two loaders are used:

- A Bootstrap loader loads the ABSolute loader and jumps to it.
- ABSolute loader loads PAL-11A output, checks for checksum errors and jumps to a user program or halts when done.

**COKE DUMP ROUTINES**—Routines are provided which dump specified ranges of core locations on paper tape in absolute format or on the teleprinter in octal.

## CHAPTER 12

# THE OPERATOR'S CONSOLE

The PDP-11 Operator's Console has been configured to achieve convenient control of the system. Through switches and keys on the console, programs or information can be manually inserted or modified. Also indicator lamps on the console face display the status of the machine, the contents of the Bus Address Register and the data at the output of the data paths.

The console is shown in Figure 12-1.

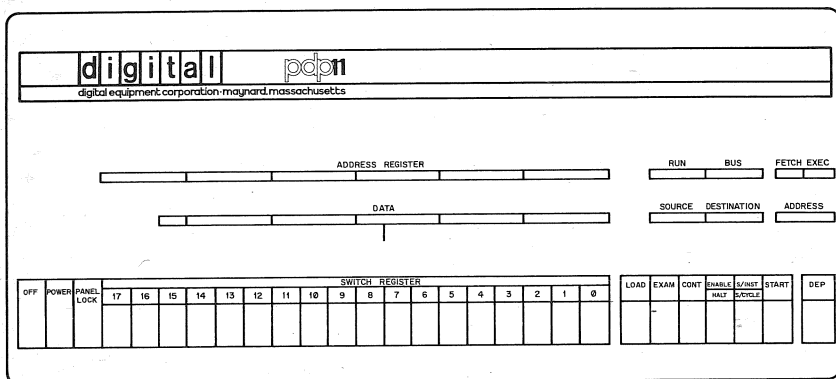


Figure 12-1

### CONSOLE ELEMENTS

The console has the following indicators and switches:

1. A bank of 8 indicators, indicating the following conditions or operations: Fetch, Execute, Bus, Run, Source, Destination and Address (2 bits).
2. An 18-bit Address Register Display
3. A 16-bit Data Register Display
4. An 18-bit Switch Register
5. Control Switches:
  - a. LOAD ADDR (Load Address)
  - b. EXAM (Examine)
  - c. CONT (Continue)
  - d. ENABLE/HALT
  - e. S/INST—S/CYCLE (Single Instruction/Single Cycle)
  - f. START
  - g. DEP (Deposit)

**INDICATOR LIGHTS**—The indicators signify specific machine functions, operations, or states. Each is defined below.

1. Fetch—indicates that the central processor is in the state of fetching an instruction.
2. Execute—indicates that the central processor is in the state of executing an instruction.

3. **Bus**—indicates that a peripheral is controlling the bus. It is lit when BBSY (Bus Busy) is asserted, unless the processor (which includes the console) is asserting BBSY.
4. **Run**—indicates that the processor is running. It monitors the control flip-flop for the internal clock.
5. **Source**—indicates that the central processor is obtaining source data except from an internal register.
6. **Destination**—indicates that the central processor is obtaining destination data (except from an internal register).
7. **Address**—identifies the source or destination address cycle of the central processor, using two lights that are decoded zero, one, two, or three. When references are made via the Unibus to the addresses, the lights tell the machine's source or destination cycle. For an internal register reference, there is a "zeroth" addressing operation.

**REGISTER DISPLAYS**—The Operator's Console has an 18-bit Address Register Display and a 16-bit Data Register Display. The Address Register Display is tied directly to the output of an 18-bit flip-flop register called the Bus Address Register. This register displays the address of data examined or deposited.

The 16-bit data register is divided on the face of the console by a line into two 8-bit bytes. This register is tied to the output of the processor data paths and will reflect the output of the processor adder.

**SWITCH REGISTER**—The PDP-11/10 and PDP-11/20 can reference  $2^{16}$  bytes addresses. However, the Unibus has expansion capability for  $2^{18}$  byte addresses. In order that the console can access the entire 18-bit address scheme, the switch register is 18 bits wide. These bits are assigned as 0 through 17. The highest two are used only as addresses. A switch in the "up" position is considered to have a "1" value and in the "down" position to have a "0" value. The condition of the 18 switches can be loaded into the bus address register or any memory location by using the appropriate control switches which are described below.

**CONTROL SWITCHES**—The switches listed in item 5 of the "Console Elements" have these specific control functions:

1. **LOAD ADDR**—transfers the contents of the 18-bit switch register into the bus address register.
2. **EXAM**—displays the contents of the location specified by the bus address register.
3. **DEP**—deposits the contents of the low 16 bits of the switch register into the address then displayed in the address register. (This switch is actuated by raising it.)
4. **ENABLE/HALT**—allows or prevents running of programs. For a program to run, the switch must be in the ENABLE position (up). Placing the switch in the HALT position (down) will halt the system.
5. **START**—starts executing a program when the ENABLE/HALT switch is in the ENABLE position. When the START switch is depressed, it asserts a system initialization signal; the system actually starts when the switch is released. The processor will start executing at the address which was last loaded by the LOAD ADDR key.
6. **CONT**—allows the machine to continue without initialization from whatever state it was in when halted.
7. **S/INST-S/CYCLE**—determines whether a single instruction or a single bus cycle is performed when the CONT switch is depressed while the machine is in the halt mode.



When the system is running a program, the LOAD ADDR, EXAM, and DEPOSIT functions are disabled to prevent disrupting the program. When the machine is to be halted, the ENABLE/HALT switch is thrown to the halt position. The machine will halt either at the end of the current instruction, or at the end of the current bus cycle, depending upon the position of the S/INST-S/CYCLE switch.

## OPERATING THE CONTROL SWITCHES

When the PDP-11 has been halted, it is possible to examine and update bus locations. To examine a specific location, the operator sets the switches of the switch register to correspond to the location's address. The operator then presses LOAD ADDR, which will transfer the contents of the switch register into the bus address register. The location of the address to be examined is then displayed in the address register display. The operator then depresses EXAM. The data in that location will appear in the data register display.

If the operator then depresses EXAM again, the bus address register will be incremented by 2 to the next word address and the new location will be examined. In the PDP-11, the bus address register will always be pointing to the data currently displayed in the data register. The incrementation occurs when the EXAM switch is depressed, and then the location is examined.

The examine function has been designed so that if LOAD ADDR and then EXAM are depressed, the address register will not be incremented. In this case, the location reflected in the address register display is examined directly. However, on the second (and successive) depressings of EXAM, the bus address register is incremented. This will continue for successive depressings as long as another control switch is not depressed.

If the operator finds an incorrect entry in the data register, he can enter new data there by putting it in the switch register and raising the DEP key. The address register will not increment when this data is deposited. Therefore, when the operator presses the EXAM key, he can examine the data he just deposited. However, when he presses EXAM again, the system will increment.

If the operator attempts to examine data from, or deposit data into, a non-existent memory location, the "time out" feature will cause an error flag. The data register will then reflect location 4, the trap location, for references to nonexistent locations. To verify this condition, the operator should try to deposit some number other than four in that location; if four is still indicated, this would indicate that either nothing is assigned to that location, or that whatever is assigned to that location is not working properly.

When doing consecutive examines or consecutive deposits, the address will increment by 2, to successive word locations. However, if the programmer is examining the fast registers (the "scratch pad" memory), the system only increments by 1. The reason for this is that once the switch register is set properly, the programmer can then use the four least significant bits of the switch register in examining fast memory registers from the front panel.

To start a PDP-11 program, the programmer loads the starting address of the program in the switch register, depresses LOAD ADDR, and after ensuring that the ENABLE/HALT switch is in the ENABLE position, depresses START. The program will start to run as soon as the START switch is released.

The Run indicator lamp is driven off the flip-flop that controls the clock. Normally, when the system is running, not only will this light be on, but the

other lights (Fetch, Execute, Source, Destination, the Address lights, and the Address and Data registers) will be flickering. If the run light is on, and none of the other indicators are flickering, the system could be executing a "wait" instruction which waits for an interrupt.

While in the halt mode, if the operator wishes to do a single instruction, he places the S/INST-S/CYCLE switch in the S/INST position and depresses CONT. When CONT is depressed, the console momentarily passes control to the processor, allowing the machine to execute one instruction before regaining control. Each time the CONT switch is depressed, the machine will execute one instruction.

Similarly, if the operator wishes to have the machine perform a single bus cycle, he places the S/INST-S/CYCLE switch in the S/CYCLE position and presses CONT. The machine will then perform one complete bus cycle and halt. The operator cannot do an examine or deposit function at the end of a single bus cycle unless the cycle ends coincidental with the end of an instruction. This prevents altering machine flow. Only when the machine is at the end of an instruction and in the halt mode can the examine or deposit functions operate.

To start the machine running its program again, the operator places the ENABLE/HALT switch in the ENABLE position, and depresses the CONT switch.

# APPENDIX A—PDP-11 INSTRUCTION REPERTOIRE

| Mnemonic                                  | Instruction Operation  | OP Code | Condition Codes ZNCV | Timing |
|---|--|---------|----------------------|--------|
| <b>DOUBLE OPERAND GROUP: OPR scr, dst</b> |  |         |                      |        |
| MOV(B)                                    | MOVe (Byte)<br>(src) → (dst)   | ·1SSDD  | ✓ ✓ — 0              | 2.3    |
| CMP(B)                                    | CoMPare (Byte)<br>(src) — (dst)                                      | ·2SSDD  | ✓ ✓ ✓ ✓              | 2.3*   |
| BIT(B)                                    | Blt Test (Byte)<br>(src) ∧ (dst)                                     | ·3SSDD  | ✓ ✓ — 0              | 2.9*   |
| BIC(B)                                    | Blt Clear (Byte)<br>~ (src) ∧ (dst) → (dst)                          | ·4SSDD  | ✓ ✓ — 0              | 2.9    |
| BIS(B)                                    | Blt Set (Byte)<br>(src) ∨ (dst) → (dst)                              | ·5SSDD  | ✓ ✓ — 0              | 2.3    |
| ADD                                       | ADD<br>(src) + (dst) → (dst)   | 06SSDD  | ✓ ✓ ✓ ✓              | 2.3    |
| SUB                                       | SUBtract<br>(dst) — (src) → (dst)                                    | 16SSDD  | ✓ ✓ ✓ ✓              | 2.3    |
| <b>CONDITIONAL BRANCHES: Bxx loc</b>      |  |         |                      |        |
| BR  | BRanch (unconditionally)<br>loc → (PC)                               | 0004XX  | — — — —              | 2.6—   |
| BNE                                       | Branch if Not Equal (Zero)<br>loc → (PC) if Z = 0                    | 0010XX  | — — — —              | 2.6—   |
| BEQ                                       | Branch if Equal (Zero)<br>loc → (PC) if Z = 1                        | 0014XX  | — — — —              | 2.6—   |
| BGE                                       | Branch if Greater or Equal (Zero)<br>loc → (PC) if N ∨ V = 0         | 0020XX  | — — — —              | 2.6—   |
| BLT                                       | Branch if Less Than (Zero)<br>loc → (PC) if N ∨ V = 1                | 0024XX  | — — — —              | 2.6—   |
| BGT                                       | Branch if Greater Than (Zero)<br>loc → (PC) if Z ∨ (N ∨ V = 0)       | 0030XX  | — — — —              | 2.6—   |
| BLE                                       | Branch if Less Than or Equal (Zero)<br>loc → (PC) if Z ∨ (N ∨ V) = 1 | 0034XX  | — — — —              | 2.6—   |
| BPL                                       | Branch if PLus<br>loc → (PC) if N = 0                                | 1000XX  | — — — —              | 2.6—   |
| BMI                                       | Branch if MInus<br>loc → (PC) if N = 1                               | 1004XX  | — — — —              | 2.6—   |
| BHI                                       | Branch if Hlgher<br>loc → (PC) if C ∨ Z = 0                          | 1010XX  | — — — —              | 2.6—   |
| BLOS                                      | Branch if LLower or Same<br>loc → (PC) if C ∨ Z = 1                  | 1014XX  | — — — —              | 2.6—   |
| BVC                                       | Branch if oVerflow Clear<br>loc → (PC) if V = 0                      | 1020XX  | — — — —              | 2.6—   |
| BVS                                       | Branch if oVerflow Set<br>loc → (PC) if V = 1                        | 1024XX  | — — — —              | 2.6—   |
| BCC                                       | Branch if Carry Clear<br>loc → (PC) if C = 0                         | 1030XX  | — — — —              | 2.6—   |
| (or BHIS)<br>BCS<br>(or BLO)              | Branch if Carry Set<br>loc → (PC) if C = 1                           | 1034XX  | — — — —              | 2.6—   |

**SUBROUTINE CALL: JSR reg, dst**

|     |  |        |   |     |
|-----|--|--------|---|-----|
| JSR | Jump to SubRoutine<br>(dst) → (tmp), (reg) ↓<br>(PC) → (reg), (tmp) → (PC) | 004RDD | — | 4.2 |
|-----|--|--------|---|-----|

**SUBROUTINE RETURN: RTS reg**

|     |   |        |   |     |
|-----|---|--------|---|-----|
| RTS | ReTurn from Subroutine<br>(reg) → PC, ↑ (reg) | 00020R | — | 3.5 |
|-----|---|--------|---|-----|

**SINGLE OPERAND GROUP: OPR dst**

|        |  |        |         |      |
|--------|--|--------|---------|------|
| CLR(B) | CLear (Byte)<br>0 → (dst)  | ·050DD | 1000    | 2.3  |
| COM(B) | COMplement (Byte)<br>~ (dst) → (dst)                             | ·051DD | ✓ ✓ 00  | 2.3  |
| INC(B) | INCrement (Byte)<br>(dst) + 1 → (dst)                            | ·052DD | ✓ ✓ — ✓ | 2.3  |
| DEC(B) | DECrement (Byte)<br>(dst) - 1 → (dst)                            | ·053DD | ✓ ✓ — ✓ | 2.3  |
| NEG(B) | NEGate (Byte)<br>~ (dst) + 1 → (dst)                             | ·054DD | ✓ ✓ ✓ ✓ | 2.3  |
| ADC(B) | ADd Carry (Byte)<br>(dst) + (C) → (dst)                          | ·055DD | ✓ ✓ ✓ ✓ | 2.3  |
| SBC(B) | SuBtract Carry (Byte)<br>(dst) - (C) → (dst)                     | ·056DD | ✓ ✓ ✓ ✓ | 2.3  |
| TST(B) | TeST (Byte)<br>0 - (dst)   | ·057DD | ✓ ✓ 00  | 2.3* |
| ROR(B) | ROtate Right (Byte)<br>rotate right 1 place with C               | ·060DD | ✓ ✓ ✓ ✓ | 2.3° |
| ROL(B) | ROtate Left (Byte)<br>rotate left 1 place with C                 | ·061DD | ✓ ✓ ✓ ✓ | 2.3° |
| ASR(B) | Arithmetic Shift Right (Byte)<br>shift right with sign extension | ·062DD | ✓ ✓ ✓ ✓ | 2.3° |
| ASL(B) | Arithmetic Shift Left (Byte)<br>shift left with lo-order zero    | ·063DD | ✓ ✓ ✓ ✓ | 2.3° |
| JMP    | JuMP<br>(dst) → (PC)   | 0001DD | —       | 1.2  |
| SWAB   | SWAp Bytes<br>bytes of a word are exchanged                      | 0003DD | ✓ ✓ 00  | 2.3  |

**CONDITION CODE OPERATORS: OPR**

1.5

Condition Code Operators set or clear combinations of condition code bits. Selected bits are set if S = 1 and cleared otherwise. Condition code bits corresponding to bits set as marked in the word below are set or cleared.

CONDITION CODE OPERATORS:

|    |  |  |   |  |  |   |  |   |  |   |   |   |   |   |   |   |
|----|--|--|---|--|--|---|--|---|--|---|---|---|---|---|---|---|
| 0  |  |  | 0 |  |  | 0 |  | 2 |  | 4 | S | N | Z | V | C |   |
| 15 |  |  |   |  |  |   |  |   |  |   | 5 | 4 | 3 | 2 | 1 | 0 |

Thus SEC = 000261 sets the C bit and has no effect on the other condition code bits (CLC = 000241 clears the C Bit)

**OPERATE GROUP: OPR**

|      |  |        |   |     |
|------|--|--------|---|-----|
| HALT | HALT<br>processor stops; (RO) and the HALT address in lights | 000000 | — | 1.8 |
| WAIT | WAIT<br>processor releases bus, waits for interrupt          | 000001 | — | 1.8 |

|       |   |               |         |        |
|-------|---|---------------|---------|--------|
| RTI   | ReTurn from Interrupt<br>↑ (PC), ↑ (PS)                       | 000002        | ✓ ✓ ✓ ✓ | 4.8    |
| IOT   | Input/Output Trap<br>(PS) ↓, (PC) ↓, (20) → (PC), (22) → (PS) | 000004        | ✓ ✓ ✓ ✓ | 8.9    |
| RESET | RESET<br>an INIT pulse is issued by the CP                    | 000005        | —       | 20 ms. |
| EMT   | EMulator Trap<br>(PS) ↓, (PC) ↓, (30) → (PC), (32) → (PS)     | 104000—104377 | ✓ ✓ ✓ ✓ | 8.9    |
| TRAP  | TRAP<br>(PS) ↓, (PC) ↓, (34) → (PC), (36) → (PS)              | 104400—104777 | ✓ ✓ ✓ ✓ | 8.9    |

## NOTATION:

### 1. for order codes

word/byte bit, set for byte (+100000)  
SS—source field,  
DD—destination field  
XX—offset (8 bit)

### 2. for operations

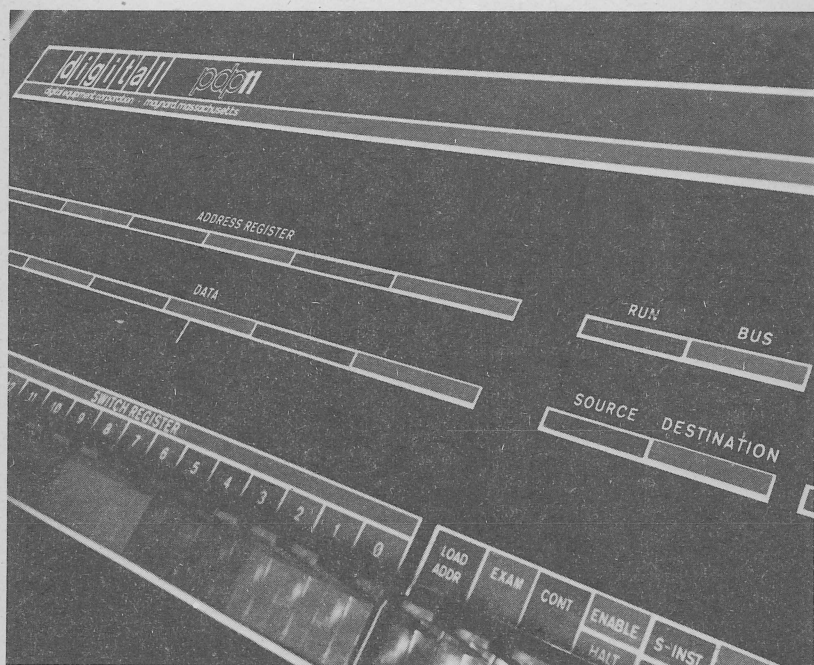
∧ and,  
v or,  
~ not,  
( ) contents of,  
⊕ XOR  
↓ "is pushed onto the processor stack"  
↑ "the contents of the top of the processor stack is popped and becomes"  
→ "becomes"

### 3. for timing

\* 0.4 μs less if not register mode  
— 0.9 μs less if conditions for branch not met  
° 1.2 μs more if addressing odd byte  
(0.6 μs additional in addressing odd bytes otherwise)

### 4. for condition codes

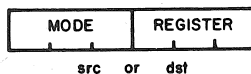
✓ set conditionally  
— not affected  
0 cleared  
1 set



The PDP-11 derives speed and memory efficiency from its wide range of addressing capabilities.

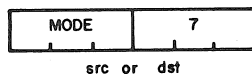
# APPENDIX B—ADDRESSING SUMMARY

## ADDRESSING MODES



### GENERAL REGISTER ADDRESSING

| Mode | Description             | Symbolic         | Timing (μs) |     |
|------|-------------------------|------------------|-------------|-----|
|      |                         |                  | src         | dst |
| 0    | register                | R                | 00          | 00  |
| 1    | register deferred       | @ R or (R)       | 1.5         | 1.4 |
| 2    | auto increment          | (R) +            | 1.5         | 1.4 |
| 3    | auto increment deferred | @ (R) +          | 2.7         | 2.6 |
| 4    | auto decrement          | — (R)            | 1.5         | 1.4 |
| 5    | auto decrement deferred | @ — (R)          | 2.7         | 2.6 |
| 6    | indexed                 | X (R)            | 2.7         | 2.6 |
| 7    | indexed deferred        | @ X (R) or @ (R) | 3.9         | 3.8 |

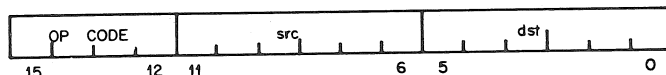


### PC REGISTER ADDRESSING

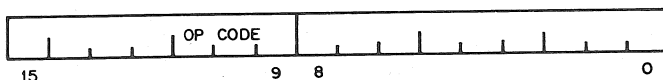
| Mode | Description       | Symbolic | Timing μs |     |
|------|-------------------|----------|-----------|-----|
|      |                   |          | src       | dst |
| 2    | immediate         | # n      | 1.5       | 1.4 |
| 3    | absolute          | @ #A     | 2.7       | 2.6 |
| 6    | relative          | A        | 2.7       | 2.6 |
| 7    | relative deferred | @ A      | 3.9       | 3.8 |

## INSTRUCTION FORMATS

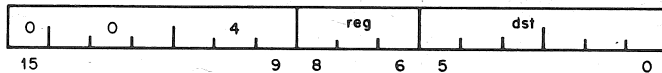
DOUBLE OPERAND GROUP: OPR src, dst



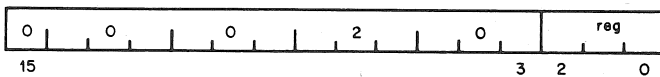
CONDITIONAL BRANCHES: Bxx loc (loc=(offset·2)+. +2)



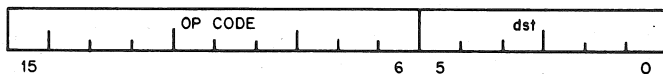
SUBROUTINE CALL: JSR reg, dst



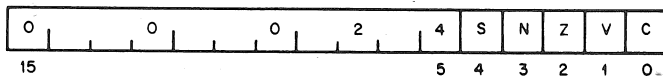
SUBROUTINE RETURN: RTS reg



SINGLE OPERAND GROUP: OPR dst



CONDITION CODE OPERATORS:





## APPENDIX C—ADDRESS MAP

|                              |  |
|------------------------------|--|
| 0                            | USER DEVICE INTERRUPT VECTOR                                   |
| 4                            | BUS ERROR, ILLEGAL INSTRUCTION, STACK OVERFLOW TRAP VECTOR     |
| 10                           | RESERVED INSTRUCTIONS TRAP VECTOR                              |
| 14                           | CODE 000003 AND TRACE TRAP VECTOR                              |
| 20                           | IOT INSTRUCTION TRAP VECTOR                                    |
| 24                           | POWER FAIL INTERRUPT VECTOR                                    |
| 30                           | EMT INSTRUCTION TRAP VECTOR                                    |
| 34                           | TRAP INSTRUCTION TRAP VECTOR                                   |
| 40                           | } SYSTEM SOFTWARE COMMUNICATION                                |
| 44                           |  |
| 50                           |  |
| 54                           |  |
| 60                           |  |
| 64                           | TELEPRINTER INTERRUPT VECTOR                                   |
|                              | TELETYPE KEYBOARD AND LOW SPEED READER INTERRUPT VECTOR        |
| 70                           | HIGH SPEED PAPER TAPE PUNCH INTERRUPT VECTOR                   |
| 74                           | HIGH SPEED PAPER TAPE READER INTERRUPT VECTOR                  |
| .                            |  |
| .                            |  |
| .                            | (additional interrupt vectors)                                 |
| .                            |  |
| .                            |  |
| 400                          |  |
| .                            |  |
| .                            |  |
| .                            | PROCESSOR STACK  |
|                              | PROGRAM AND DATA   |
|                              | RESIDENT SYSTEM SOFTWARE                                       |
| .                            |  |
| .                            |  |
| .                            | (ABSOLUTE LOADER, BOOTSTRAP, I/O EXECUTIVE)                    |
| (end of implemented storage) |  |
| 160000                       |  |
| .                            |  |
| .                            | SMALL READ-ONLY STORAGE UNITS                                  |
| .                            |  |
| .                            |  |
| .                            |  |
| .                            |  |
| .                            | OTHER PERIPHERAL DEVICE REGISTERS                              |
| .                            |  |
| .                            |  |
| .                            |  |
| 177550                       | HIGH SPEED READER AND PUNCH DEVICE STATUS AND BUFFER REGISTERS |
| .                            |  |
| .                            |  |
| .                            |  |

177560 TELETYPE KEYBOARD AND PUNCH DEVICE STATUS AND BUFFER  
REGISTER

.  
.  
.  
.  
.

177576

177600

.

177677

177700

.

177776

} RESERVED FOR EXPANSION OF PROCESSOR REGISTERS

} GENERAL REGISTERS R0 — R7

} CENTRAL PROCESSOR STATUS REGISTER (PS)

## APPENDIX D—UNIBUS OPERATIONS

There are six bus operations: four to effect data transfers, one to transfer bus control, and one to effect a program interrupt. This appendix describes the signal interaction on the Unibus to perform these six operations.

### DATA TRANSFERS

The four data transfers use the C lines coded as follows:

| C1 | C0 |                      |
|----|----|----------------------|
| 0  | 0  | DATI-DATa In         |
| 0  | 1  | DATIP-DATa In, Pause |
| 1  | 0  | DATO-DATa Out        |
| 1  | 1  | DATOB-DATa Out, Byte |

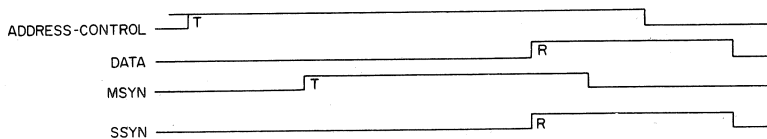
**DATI AND DATIP**—These two bus operations transfer data from a slave whose address is specified by  $A < 17:01 >$  into the master. Both transfers are made in words on  $D < 15:00 >$ . In destructive read-out devices, DATI commands a read-restore operation, while DATIP commands a read-pause operation and the setting of a pause flag. DATIPs are to be followed by a DATO or DATOB to effect a read-modify-write data exchange. In non destructive read-out devices, DATI and DATIP are treated identically. The sequence of operations is as follows:

1. Master puts address on A, 0 or 1 on C, and waits 150 nanoseconds. (75 nanoseconds for deskewing address + 75 nanoseconds for address decoding).
2. Master asserts MSYN.
3. Slave decodes address, sees 0 or 1 on C, and MSYN and begins read cycle (flip-flop register would simply gate flop outputs to bus).
4. Slave completes read cycle, outputs data to D lines, and asserts SSYN. If the slave is a destructive read-out device, it now restores data on a DATI: it sets a pause flag on a DATIP.

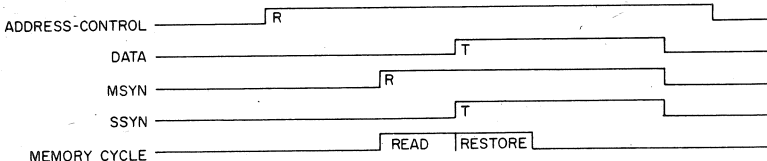
Figure D-1 shows the signals for a DATI operation.

#### DATI

##### SIGNALS AT MASTER



##### SIGNALS AT SLAVE



T = SIGNAL AS TRANSMITTED  
R = SIGNAL AS RECEIVED

Figure D-1 DATI Operation

5. Master sees SSYN and waits 75 nanoseconds, minimum (data deskewing + internal gating deskewing).
6. Master strobes data, drops MSYN, and waits 75 nanoseconds (deskew address).
7. Master drops A and C and waits for SSYN to fall.
8. Slave sees MSYN fall and drops SSYN and D lines.
9. Master sees SSYN fall, signaling end of bus operation.

**NOTES:**

1. Step 1 of the next data transfer may begin at step 7 of the current DATI or DATIP.
2. Step 2 of the next data transfer may begin at step 9 of the current DATI or DATIP.

**DATO AND DATOB**—These two bus operations transfer data out of the master to the slave. DATO is used to transfer a word to the address specified by A < 17:01 >. The slave ignores A00 and the data appears on D < 15:00 >. DATOB is used to transfer a byte to the address specified by A < 17:00 >. A00 = 0 indicates the low byte and data appears on D < 07:00 >; A00 = 1 indicates high byte and data appears on D < 15:08 >. The sequence of operation is as follows:

1. Master puts address on A, data on D, 2 or 3 on C, and waits 150 nanoseconds (75 nanoseconds for deskewing address + 75 nanoseconds for address decoding).
2. Master asserts MSYN.
3. Slave decodes address, sees 2 or 3 on C and MSYN and strobes in word or byte. When slave has taken data, it asserts SSYN. If the slave is a destructive read-out device and its pause flag is set (by DATIP), slave begins write cycle; if not, slave must first do a read cycle to clear the memory cell and then a write.
4. Master sees SSYN and drops MSYN and waits 75 nanoseconds (deskew address).
5. Master drops A, D, and C, and waits for SSYN to fall.
6. Slave sees MSYN fall and drops SSYN.
7. Master sees SSYN fall, signaling end of bus operation.

Figure D-2 shows the signals for a DATO operation.

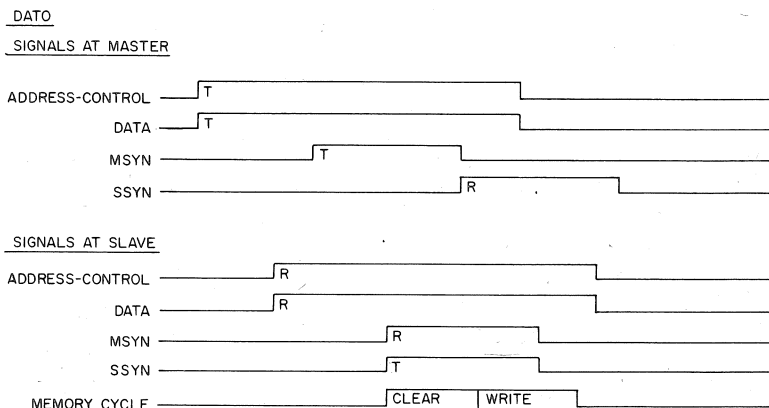


Figure D-2 DATO Operation

# NOTES:

1. Step 1 of the next data transfer may begin at step 5 of the current DATO or DATOB.
2. Step 2 of the next data transfer may begin at step 7 of the current DATO or DATOB.

## PTR-PRIORITY TRANSFER

This bus operation is used to pass control of the bus from one master to another. The steps which follow are performed simultaneously with the data transfers:

0. Current master device always has BBSY asserted.
1. Requesting device asserts its assigned BR line.
2. Processor sees BR asserted, determines which BR is highest, and asserts the corresponding BG line if the processor's current priority level allow that level of bus request.
3. Each device that receives the BG passes it on to the next device unless it itself is requesting.
4. The BG is propagated along the priority chain until it reaches the first requesting device. This device becomes selected as next bus master and does not allow the BG to pass to succeeding devices.
5. The selected device asserts SACK and drops its BR, and waits for BBSY, BG, and SSYN to drop.
6. The processor sees SACK and drops BG.
7. The device which is current master completes its data transfers, drops BBSY, and ceases to be bus master.
8. The selected device sees BG, BBSY, and SSYN drop, becomes bus master, asserts BBSY, drops SACK, and begins data transfers.
9. New master relinquishes bus control, either to the processor or to a requesting device, by dropping BBSY at the end of its last bus operation. This is termed a passive release of bus control.

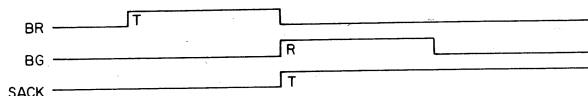
# NOTES:

1. NPR bus requests are handled as above.
2. Processor defers action on BR  $\langle 7:4 \rangle$  until last bus cycle of an instruction execution or interrupt sequence, NPR is acted upon immediately.
3. Processor becomes bus master and asserts BBSY whenever it sees BBSY = 0 and no other device has been selected or is being selected as next bus master.
4. Processor will not execute step 2 if SACK is asserted. See note 2 under INTR.

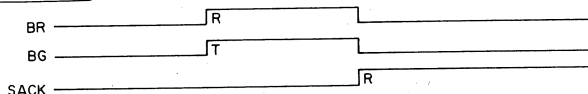
Figure D-3 shows the signals for a PTR operation.

## PTR

### SIGNALS AT DEVICE



### SIGNALS AT PROCESSOR



T = SIGNAL AS TRANSMITTED  
R = SIGNAL AS RECEIVED

Figure D-3 PTR Operation

## INTR—INTerRupt

This bus operation is initiated by a master immediately after receiving bus control to effect a program interrupt in the processor. It proceeds as follows:

0. Device has become bus master via PTR and BBSY is asserted.
1. Master puts interrupt vector address on D and asserts INTR.
2. Processor sees INTR and waits 75 nanoseconds (deskew data).
3. Processor strobes data and asserts SSYN.
4. Master sees SSYN, drops INTR, D, and BBSY. The master has now relinquished bus control directly to the processor. The INTR sequence is termed an active release of bus control.\*
5. Processor sees INTR drop and drops SSYN and enters interrupt sequence to update PC and PS.

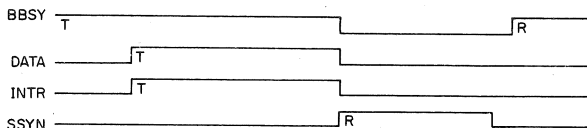
### NOTES:

1. Step 1 must be made simultaneously with step 8 of PTR; that is, SACK cannot be dropped until INTR is asserted.
2. When the processor sees SACK drop, it waits 75 nanoseconds (deskew). If, at that time, INTR = 1, the processor issues no BG's until the interrupt sequence is complete.

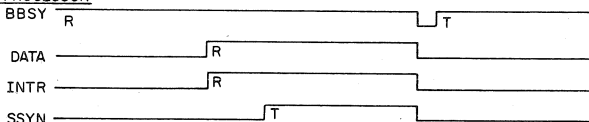
Figure D-4 shows the signals for the INTR operation.

### INTR

#### SIGNALS AT MASTER



#### SIGNALS AT PROCESSOR



T = SIGNAL AS TRANSMITTED

R = SIGNAL AS RECEIVED

Figure D-4 INTR Operation

## GENERAL NOTES ON THE BUS OPERATIONS

1. A master device doing a read-modify-write operation must keep bus control BBSY asserted for both bus transactions (both the DATIP and the DATO or DATOB). This is the one case where an NPR request will not be honored between bus transactions.
2. A device becomes master by the PTR operation. If the request for bus control was made on the NPR line, bus control must be released passively (by dropping BBSY). Bus control is then passed either back to the processor to execute the next bus cycle of the instruction or to another device requesting on the NPR line. If a device becomes master via a BR request line, control may be passed actively back to the processor by using the INTR operation or passively (by drop-

ping BBSY). If control is given up actively, only NPR requests will be honored during the interrupt sequence of updating the PC and PS. If control is given up passively, control may pass either to the processor to fetch the next instruction or to an NPR requesting device.



The PDP-11 provides Direct Device Addressing. All memory and devices on the Unibus are directly addressable and may be operated upon by all computer instructions. Direct device to device transfers are possible.



## DIGITAL EQUIPMENT CORPORATION



## WORLD-WIDE SALES AND SERVICE

## MAIN OFFICE AND PLANT

146 Main Street, Maynard, Massachusetts 01754 • Telephone: From Metropolitan Boston: 646-8500 • Elsewhere: (617) 897-5111 • TWX: 710-347-0212 Cable: Digital Mayn. Telex: 94-8457

## UNITED STATES

## NORTHEAST

## NORTHEAST OFFICE:

15 Lunda Street, Waltham, Massachusetts 02154  
Telephone: (617) 891-1030 & 1033

## WALTHAM OFFICE:

15 Lunda Street, Waltham, Massachusetts 02154  
Telephone: (617) 891-1030 & 1033

## CAMBRIDGE/BOSTON OFFICE:

889 Main Street, Cambridge, Massachusetts 02139  
Telephone: (617) 491-6130 TWX: 710-320-1167

## ROCHESTER OFFICE:

130 Allens Creek Road, Rochester, New York 14618  
Telephone: (716) 461-1700 TWX: 510-253-3078

## CONNECTICUT OFFICE:

1 Prestige Drive, Meriden, Connecticut 06450  
Telephone: (203) 237-8441 TWX: 710-461-0054

## MID-ATLANTIC—SOUTHEAST

## MID-ATLANTIC OFFICE:

U.S. Route 1, Princeton, New Jersey 08540  
Telephone: (609) 452-9150 TWX: 510-685-2338

## NEW YORK OFFICE:

95 Cedar Lane, Englewood, New Jersey 07631  
Telephone: (201) 871-4944 (212) 594-6955, (212) 736-0447  
TWX: 710-991-9721

## NEW JERSEY OFFICE:

1259 Route 46, Parsippany, New Jersey 07054  
Telephone: (201) 385-3500 TWX: 610-587-8919

## PRINCETON OFFICE:

Route One and Emmons Drive,  
Princeton, New Jersey 08540  
Telephone: (609) 452-9150 TWX: 510-685-2337

## LONG ISLAND OFFICE:

1919 Middle Country Road,  
Centereach, L.I., New York 11720  
Telephone: (516) 585-5410 TWX: 510-228-6505

## PHILADELPHIA OFFICE:

1100 West Valley Road, Wayne, Pennsylvania 19087  
Telephone: (215) 867-1405 TWX: 510-688-4461

## WASHINGTON OFFICE:

Executive Building  
7100 Baltimore Ave., College Park, Maryland 20740  
Telephone: (301) 779-1100 TWX: 710-628-9662

## MID-ATLANTIC—SOUTHEAST (cont.)

## CHAPEL HILL OFFICE:

2704 Chapel Hill Boulevard  
Durham, North Carolina 27707  
Telephone: (919) 489-3347 TWX: 510-927-0912

## HUNTSVILLE OFFICE:

Suite 41 — Holiday Office Center  
3322 Memorial Parkway S.W., Huntsville, Ala. 35801  
Telephone: (205) 881-7730 TWX: 810-726-2122

## ORLANDO OFFICE:

Suite 232, 8890 Lake Ellenor Drive, Orlando, Fla. 32809  
Telephone: (305) 851-4450 TWX: 810-850-0180

## ATLANTA OFFICE:

Suite 116, 1700 Commerce Drive, N.W.,  
Atlanta, Georgia 30318  
Telephone: (404) 351-2822 TWX: 810-751-3251

## KNOXVILLE OFFICE:

5731 Lyons View Dr., S.W., Knoxville, Tenn. 37919  
Telephone: (615) 588-6511 TWX: 810-583-0123

## CENTRAL

## CENTRAL OFFICE:

1890 Frontage Road, Northbrook, Illinois 60062  
Telephone: (312) 498-2560 TWX: 910-686-0555

## PITTSBURGH OFFICE:

400 Penn Center Boulevard,  
Pittsburgh, Pennsylvania 15235  
Telephone: (412) 243-8500 TWX: 710-797-3657

## CHICAGO OFFICE:

1850 Frontage Road, Northbrook, Illinois 60062  
Telephone: (312) 498-2560 TWX: 910-686-0555

## ANN ARBOR OFFICE:

230 Huron View Boulevard, Ann Arbor, Michigan 48103  
Telephone: (313) 761-1150 TWX: 810-223-6053

## INDIANAPOLIS OFFICE:

21 Beachway Drive — Suite G  
Indianapolis, Indiana 46224  
Telephone: (317) 243-8341 TWX: 810-341-3436

## MINNEAPOLIS OFFICE:

15015 Minnetonka Industrial Road  
Minnetonka, Minnesota 55343  
Telephone: (612) 935-1744 TWX: 910-576-2818

## CLEVELAND OFFICE:

Park Hill Bldg., 3510 Euclid Ave.  
Willoughby, Ohio 44094  
Telephone: (216) 946-8484 TWX: 810-427-2608

## INTERNATIONAL

## CANADA

## CANADIAN OFFICE:

Digital Equipment of Canada, Ltd.  
150 Rossmore Street, Carleton Place, Ontario  
Telephone: (613) 257-2615 TWX: 610-581-1651

## OTTAWA OFFICE:

Digital Equipment of Canada, Ltd.  
120 Holland Street, Ottawa 3, Ontario  
Telephone: (613) 725-1193 TWX: 610-582-8907

## TORONTO OFFICE:

Digital Equipment of Canada, Ltd.  
230 Lakeshore Road East, Port Credit, Ontario  
Telephone: (416) 278-6111 TWX: 610-492-4306

## MONTREAL OFFICE:

Digital Equipment of Canada, Ltd.  
640 Cathcart Street, Suite 205, Montreal, Quebec  
Telephone: (514) 861-6394 TWX: 610-421-3690

## EDMONTON OFFICE:

Digital Equipment of Canada, Ltd.  
351-103 Street  
Edmonton, Alberta, Canada  
Telephone: (403) 434-9333 TWX: 610-831-2248

## EUROPEAN HEADQUARTERS

Digital Equipment Corporation International-Europe  
81 Route De L'Aire  
1227 Carouge / Geneva, Switzerland  
Telephone: 42 79 50 Telex: 22 683

## GERMANY

COLOGNE OFFICE:  
Digital Equipment GmbH  
5 Koeln, Biemarckstrasse 7, West Germany  
Telephone: 82 21 81 Telex: 841-885-2269  
Telegram: Flip Chip Koeln

## MUNICH OFFICE:

Digital Equipment GmbH  
8000 Muenchen 19, Leonordstrasse 58  
Telephone: 516 30 54 TELEX: 841 524226

## ENGLAND

## READING OFFICE:

Digital Equipment Co. Ltd.  
Arkwright Road, Reading, Berkshire, England  
Telephone: Reading RG1 1P1 Telex: 84327

## MANCHESTER OFFICE:

Digital Equipment Co. Ltd.  
13/15 Upper Precinct, Walkden  
Manchester, England M20 5AZ  
Telephone: 061-750-4591/2 Telex: 658666

## LONDON OFFICE:

Digital Equipment Co. Ltd.  
Bilton House, Uxbridge Road, Ealing, London W.5.  
Telephone: 01-579-2761 Telex: 84327

## FRANCE

## PARIS OFFICE:

Equipment Digital S.A.R.L.  
233 Rue de Charenton, Paris 12, France  
Telephone: 344-76-07 Telex: 21339

## BENELUX

## THE HAGUE OFFICE:

(serving Belgium, Luxembourg, and The Netherlands)  
Koningsingracht N.V.  
Koningsingracht 65, The Hague, Netherlands  
Telephone: 635960 Telex: 52533

## SWEDEN

## STOCKHOLM OFFICE:

Digital Equipment Aktiebolag  
Vretensgatan 2, S-171 54 Solna, Sweden  
Telephone: 08 98 13 90 TELEX: 170 50 Digital S  
Cable: Digital Stockholm

## SWITZERLAND

## SWITZERLAND OFFICE:

Digital Equipment Corporation S.A.  
81 Route De L'Aire  
1227 Carouge / Geneva, Switzerland  
Telephone: 42 79 50 Telex: 22 683

## CENTRAL (cont.)

## ST. LOUIS OFFICE:

Suite 110, Progress Pky., Maryland Heights,  
Missouri 63042  
Telephone: (314) 872-7520 TWX: 910-764-0831

## DAYTON OFFICE:

3101 Kettering Blvd., Dayton, Ohio 45439  
Telephone: (513) 259-7377 TWX: 810-459-1676

## DALLAS OFFICE:

1625 W. Mockingbird Lane, Suite 309  
Dallas, Texas 75235  
Telephone: (214) 638-4880

## HOUSTON OFFICE:

3417 Milam Street, Suite A, Houston, Texas 77002  
Telephone: (713) 524-2961 TWX: 910-881-1651

## WEST

## WESTERN OFFICE:

560 San Antonio Road, Palo Alto, California 94305  
Telephone: (415) 328-0400 TWX: 910-373-1266

## ANAHEIM OFFICE:

801 E. Ball Road, Anaheim, California 92805  
Telephone: (714) 776-8832 or (213) 625-7659  
TWX: 910-591-1189

## WEST LOS ANGELES OFFICE:

2002 Cotner Avenue, Los Angeles, California 90025  
Telephone: (213) 473-3791 TWX: 910-342-6999

## SAN FRANCISCO OFFICE:

560 San Antonio Road, Palo Alto, California 94305  
Telephone: (415) 328-0400 TWX: 910-373-1266

## ALBUQUERQUE OFFICE:

5333 Indian School Road, N.E.  
Albuquerque, N.M. 87110  
Telephone: (505) 299-5411 TWX: 910-989-0614

## DENVER OFFICE:

2305 South Colorado Blvd., Suite #5  
Denver, Colorado 80222  
Telephone: 303-757-3332 TWX: 910-931-2650

## SEATTLE OFFICE:

1521 130th N.E., Bellevue, Washington 98004  
Telephone: (206) 454-4058 TWX: 910-433-2306

## SALT LAKE CITY OFFICE:

431 South 3rd East, Salt Lake City, Utah 84111  
Telephone: (801) 328-9838 TWX: 910-925-5834

## ITALY

## MILAN OFFICE:

Digital Equipment S. p. A.  
Corso Garibaldi, 49, 20121 Milano, Italy  
Telephone: 872 748, 872 694, 872 394 Telex: 33615

## AUSTRALIA

## SYDNEY OFFICE:

Digital Equipment Australia Pty. Ltd.  
75 Alexander Street, Crows Nest, N.S.W. 2065, Australia  
Telephone: 439-2566 Telex: AA20740

## MELBOURNE OFFICE:

Digital Equipment Australia Pty. Ltd.  
60 Park Street, South Melbourne, Victoria, 3205  
Telephone: 69-6142 Telex: AA30700

## WESTERN AUSTRALIA OFFICE:

Digital Equipment Australia Pty. Ltd.  
643 Murray Street  
West Perth, Western Australia 6005  
Telephone: 21-4993 Telex: AA92140

## BRISBANE OFFICE:

Digital Equipment Australia Pty. Ltd.  
139 Marvale Street, South Brisbane  
Queensland, Australia 4101  
Telephone: 44047 Telex: AA40616

## JAPAN

## TOKYO OFFICE:

Rikeli Trading Co., Ltd. (sales only)  
Kozato-Kaikan Bldg.  
No. 18-14, Nishitohmabashi 1-chome  
Minato-Ku, Tokyo, Japan  
Telephone: 5915246 Telex: 7814208

## Digital Equipment Corporation International

(engineering and services)  
Fukuyoshio Building, No. 2-6, Roppongi 2-Chome,  
Minato-Ku, Tokyo  
Telephone No. 585-3624 Telex No.: 0242-2650



