CHAPTER 4
PDP-8/E PROGRAMMING SYSTEMS

GENERAL

This chapter deals with the concepts required to program the PDP-8/E and identifies the system programs available to the user. Two handbooks, INTRODUCTION TO PROGRAMMING and PROGRAMMING LANGUAGES, provide a more detailed treatment and description of the commonly used programming languages and programming systems.

The chapter is divided into 2 sections. Section 1 provides basic programming guidelines and section 2 identifies the various programming systems and commonly used languages available to the user.

SECTION 1

PDP-8/E PROGRAMMING FUNDAMENTALS

Organization of the standard core memory or any 4096-word field of extended memory is summarized as follows:

Total locations (decimal)	0-4095 or 4096
Total addresses (octal)	0-7777 or 10,000
Number of pages (decimal)	0-31 or 32
Page designations (octal)	0-37 or 40
Number of locations per page (decimal)	0-127 or 128
Addresses within a page (octal)	0-177 or 200

Routines using 128 instructions or less can be written in one page using direct addresses for looping and indirect addresses for data stored in other pages. When planning the location of instructions and data in core memory, the following locations are reserved for special purposes:

Address	Purpose
0 (octal)	Stores the contents of the program counter following a program interrupt.
1 (octal)	Stores the first instruction to be executed following a program interrupt.

10 (octal)—17 (octal) Auto-indexing.

MEMORY ADDRESSING

The programmer has 4096 (decimal) locations which he may address. However, as illustrated in Figure 3-2 of Chapter 3, when an instruction is fetched from memory, only bits 5 through 11 contain the address of the data. Addressing is accomplished using octal notation. Therefore the 4096 possible locations require addresses in octal from 0000 to 7777. This means that a total of 12 bits is required to specify an absolute address. So that all locations may be addressed as efficiently as possible, memory is addressed in terms of pages with a coding scheme that allows easy access to any one of the 10,000 octal locations. The page addressing scheme is illustrated in Figure 4-1 which shows the relationship of the 40 octal pages with the 10,000 octal locations. The programmer is interested in only three pages in memory at any one time:

a. The current page
b. Page 0
c. A location on other than the current page or page 0.

Page 0 is used to store commonly used operands and off-page pointers. For instance, the location of an indirect address used by instructions is usually on Page 0.

ABSOLUTE ADDR. (OCTAL)	CORE MEMORY PAGE (OCTAL)	PAGE ADDR. (OCTAL)
7777 7600	37	177 000
7577 7400	36	177 000
7377 7200	35	177 000
7177 7000	34	177 000
6777 6600	33	177 000
6577 6400	32	177 000
6377 6200	31	177 000
6177 6000	30	177 000
5777 5600	27	177 000
5577 5400	26	177 000
5377 5200	25	177 000
5177 5000	24	177 000
4777 4600	23	177 000
4577 4400	22	177 000
4377 4200	21	177 000
4177 4000	20	177 000
3777 3600	17	177 000
3577 3400	16	177 000
3377 3200	15	177 000
3177 3000	14	177 000
2777 2600	13	177 000
2577 2400	12	177 000
2377 2200	11	177 000
2177 2000	10	177 000
1777 1600	7	177 000
1577 1400	6	177 000
1377 1200	5	177 000
1177 1000	4	177 000
0777 0600	3	177 000
0577 0400	2	177 000
0377 0200	1	177 000
0177 0000	0	177 000

Figure 4-1 Memory Addressing Scheme

The format of the Central Processor (CPMA) Memory Address Register establishing the memory page must first be considered. The MA register is an unequally divided 12-bit register in which the least significant bits of the MA (bits 5-11) are called the page address bits and the most significant bits (bits 0-4) are called the page bits. The 12 bits of the Memory Address are established by the program counter (PC) and are re-established each time the PC loads an absolute address for the next instruction. Bits 0-4 are used to establish the memory page as shown in Figure 4-2. Because the pages to be addressed include pages 0-37 (octal), only five bits are required. The first two bits represent numbers from 0 to 3 and the next three represent numbers from 0 to 7. Because the locations to be addressed on any given page include locations 0-177 (octal), only seven bits are required to specify any one location. Bit 5 represents an octal 1 or 0; bits 6, 7, and 8 represent the second octal digit from 0 to 7; bits 9, 10, and 11 represent the last octal digit from 0 to 7. Thus, on the example shown in Figure 4-2, the MA register is addressing page 5, location 73 (absolute address 1273).

When the user first receives his PDP-8/E, he should assume that it has no information content in its memory. Before he can load instructions into memory, he must first perform the initializing and loading procedures described in Chapter 2. The following discussion assumes that the preliminary procedures have been completed and that the programmer now wants to load into core those instructions which will be called upon after a program has been written. His main concern is to decide in which memory locations he desires to place his instructions and in which locations he wishes to place the corresponding data.

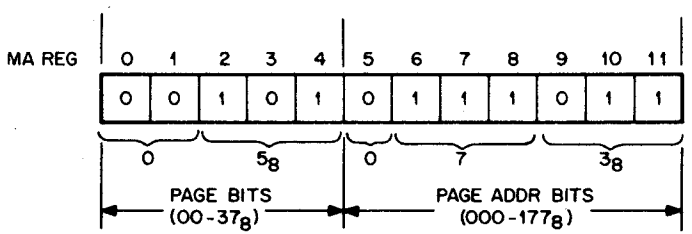

Figure 4-2 Format Establishing Memory Page

Initially, the programmer must load the Central Processor Memory Address Register with an address and then deposit a 12-bit instruction word in the format shown in Figure 4-3.

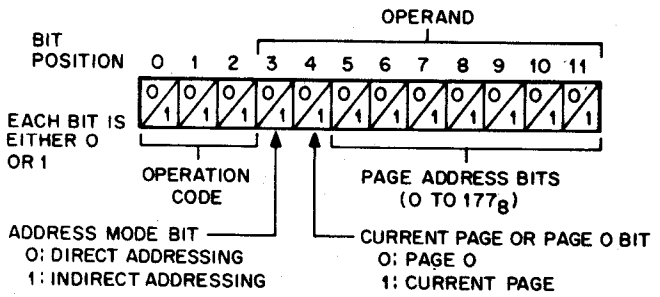

Figure 4-3 Format of a Memory Reference Instruction

The first three bits contain the instruction operation code and have nothing to do with addressing. The last seven bits address the location (from 0 to 177 octal), which will contain either data or a 12-bit address. Those address bits are located in the Memory Buffer Register, and are ineffective until bits 3 and 4 are decoded, at which time the address bits are transferred from the Memory Buffer to the Central Processor Memory Address Register. The page address (the first five bits of the MA register) is determined by whether bit 4 is a 0 or 1 (see Figure 4-4).

Where to place the instruction word or Data word is a very important consideration. At this point, the programmer has three choices in the location of the data:

a. the current page (that page containing the Instruction)
b. page 0
c. a page other than page 0 or the current page.

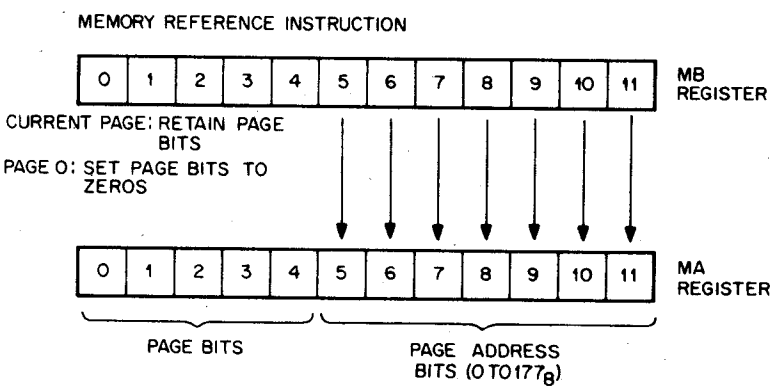

Figure 4-4 Transfer of Address Portion of MRI into MA Register

Current Page—If the programmer desires the data to be located in the current page, he must make bit 4 a 1 in the original instruction word. The logic within the processor causes the first five bits of the MA to remain, and transfers the last seven bits of the MB (the new address within a page) to the last seven bits of the MA Register. This method of updating the MA Register is illustrated in Figure 4-4.

Page 0—Page 0 is commonly used to store operands or address of operands or routines. The programmer must set bit 4 word to 0. The logic within the processor then places all zeros in MA bits 0 through 4 and transfers the content of the last 7 bits of the MB register to the last 7 bits of the MA Register. Thus, the page address is now page 0 and the address within page 0 is some address between 0 and 177 octal. This is illustrated in Figure 4-4.

Addressing A Page Other Than the Current Page or Page 0—The programmer may address a page other than the current page or page 0 by placing a 1 in bit 3 of the original instruction word. As before, the computer then goes to an address on the current page or on page 0, depending on the state of bit 4. The logic within the processor responds to bit 3 being a 1 by going into a defer state for a new address. This procedure is called "Indirect Addressing."

INDIRECT ADDRESSING

In the preceding section, the method of directly addressing 400(octal) memory locations by an MRI was described—namely those on page 0 and those on the current page. This section describes the method for addressing the other 7400(octal) memory locations. Bit 3 of an MRI designates the address mode. When bit 3 is a 0, the operand is a direct address. When bit 3 is a 1, the operand is an indirect address. An indirect address (pointer address) identifies the location that contains the desired address (effective address). To address a location that is not directly addressable, the absolute address of the desired location is stored in one of the 400(octal) directly addressable locations (pointer address); the pointer address is written as the operand of the MRI; and the letter I is written between the mnemonic and the operand. (During assembly, the presence of the I results in bit 3 of the MRI being set to 1.) Upon execution, the MRI will operate on the contents of the location identified by the address contained in the pointer location.

The two examples in Figure 4-5 illustrate the difference between direct addressing and indirect addressing. The first example shows a TAD instruction that uses direct addressing to get data stored on page 0 in location 50; the second is a TAD instruction that uses indirect addressing, with a pointer on page 0 in location 50, to obtain data stored in location 1275. (When references are made to them from various pages, constants and pointer addresses can be stored on page 0 to avoid the necessity of storing them on each applicable page.) The octal value 1050, in the first example, represents direct addressing (bit 3 = 0); the octal value 1450, in the second example, represents indirect addressing (bit 3 = 1). Both examples assume that the accumulator has previously been cleared.

Location	Content	
200	TAD 50	(TAD 50 = 1050_8)
.	.	Instruction
.	.	
50	1275	Data (Number) To Be Acted Upon By
.	.	Instruction Address
.	.	
1275	20	(Content of location 1275 is not used in the execution of the instruction in location 200.)

NOTE: AC = 1275 after executing the instruction in location 200

Location	Content	
200	TAD I 50	(TAD I 50 = 1450_8)
.	.	
.	.	Designates Indirect Addressing
.	.	Instruction
50	1275	Pointer Address
.	.	
.	.	
1275	20	Data (Number) To Be Acted Upon By Instruction Effective Address

NOTE: AC = 20 after executing the instruction in location 200.

Figure 4-5 Comparison of Direct and Indirect Addressing

The following three examples illustrate some additional ways in which indirect addressing can be used. As shown in example 1, indirect addressing makes it possible to transfer program control from off page 0 (or any other page) to any desired memory location. (Similarly, indirect addressing makes it possible for other memory reference instructions to address any of the 4,096(10) memory locations.) Example 2 shows a DCA instruction that uses indirect addressing with a pointer on the current page. The pointer in this case designates a location off the current page (location 227) in which the data is to be stored. (A pointer address is normally stored on the current page when all references to the designated location are from the current page.) Indirect addressing provides the means for returning to a main program from a subroutine, as shown in example 3. Indirect addressing is also effectively used in manipulating tables of data.

EXAMPLE 1

Location	Content	
75	JMP I 100	(JMP I 100 = 5500(octal))
.	.	
.	.	Designates Indirect Addressing Instruction

100	6000	Pointer Address
.	.	
.	.	
.	.	
6000	DCA 6100	Next Instruction To Be Executed
.	.	
.	.	

NOTE: Execution of the instruction in location 75 causes program control to be transferred to location 6000, and the next instruction to be executed is the DCA 6100 instruction.

EXAMPLE 2

Location	Content	
450	DCA I 577	(DCA I 577 = 3777(octal)) Designates Indirect Addressing Instruction
577	277	Pointer Address
.	.	
.	.	
227	nnnn	Data (Number) Stored By Instruction (Effective Address)

NOTES: 1. Memory Location 577 is location 177 of current page. Execution of the instruction in location 450 causes the contents of the accumulator to be stored in location 227.

EXAMPLE 3

Location	Content	
207	JMS I 70	(JMS I 70 = 4470(octal))
210	TAD 250	(The next instruction to be executed upon return from the subroutine.)
.	.	
.	.	
70	2000	(Starting address of the subroutine stored here.)
.	.	
.	.	
2000	aaaa	(Return address stored here by JMS instruction.)
2001	i i i	(First instruction of subroutine.)
.	.	
.	.	
2077	JMP I 2000	(Last instruction of subroutine.)

NOTES: 1. Execution of the instruction in location 207 causes the address 210 to be stored in location 2000 and the instruction in location 2001 to be executed next. Execution of the subroutine proceeds until the last instruction (JMP I 2000) causes control to be transferred back to the main program, continuing with the execution of the instruction stored in location 210.

2. A JMS instruction that uses indirect addressing is useful when the subroutine is too large to store on the current page.
3. Storing the pointer address on page 0 enables instructions on various pages to have access to the subroutine.

PROGRAMMING OPERATIONS

The programmer can make use of any combination of instructions. The following sections describe the more common programming operations.

STORING AND LOADING

Data is stored in any core memory location by use of the DCA (Deposit & Clear AC) instruction. This instruction clears the AC to simplify loading of the next data. If the data deposited is required in the AC for the next program operation, the DCA must be followed by a TAD for the same address. All loading of core memory information into the AC is accomplished by means of the TAD instruction.

The DCA instruction stores the contents of the AC in the referenced location, destroying the original contents of the location. The AC is then set to all zeroes. The following example shows the contents of the accumulator, link, and location 225 before and after executing the instruction DCA 225.

DCA 225

	AC	Link	Loc. 225
Before Execution	1234	1	7654
After Execution	0000	1	1234

The following facts should be kept in mind when using the DCA instruction:

a. The state of the link bit is not altered.
b. The AC is cleared.
c. The original contents of the addressed location are replaced by the contents of the AC.

PROGRAM CONTROL

The Program Counter is used to direct the processor to the next address of the next instruction to be fetched. Therefore, the content of the PC Register is always one more than the content of the Central Processor Memory Address (CPMA) Register. When an instruction has been completed and the processor is ready to go into a new fetch, the content of the Program Counter is transferred into the CPMA Register and the Program Counter with its original address is incremented by +1, thereby pointing to the next sequential address. This procedure is called Program Control because it directs the processor to the next instruction. Because this rigid sequence of instructions is not always desirable for practical programming, the PDP-8/E provides a means of jumping out of this sequence to transfer Program Control from one sequence of instructions to another or to allow the processor to enter a subroutine which is itself a sequence of instructions and re-enter the main program when the subroutine has been completed.

Transfer of program control to any core memory location uses the JMP or JMS instructions. The JMP I and JMS I (indirect address, bit 3 = 1) are used to transfer program control to any location in core memory which is not in the current page or page 0.

The JMS Y is used to enter a subroutine which starts at location Y +1 in the current page or page 0. The contents of the PC are stored in the specified address Y, and address Y + 1 is transferred into the PC. Subroutines or other pages may be entered via an indirect JMS. To exit a subroutine, the last instruction is a JMP I Y, which returns program control to the location stored in Y.

The JMP instruction loads the effective address of the instruction into the program counter, thereby changing the program sequence since the PC specifies the next instruction to be performed. In the following example, execution of the instruction in location 250 (JMP 300) causes the program to jump over the instructions in locations 251 through 277 and immediately transfer control to the instruction in location 300.

```
Location    Content
   250      JMP 300     (This instruction transfers program
    .           .       control to location 300.)
    .           .
   300      DCA 300
    .           .
```

NOTE: The JMP instruction does not affect the contents of the AC or link.

A program written to perform a specific operation often includes sets of instructions which perform intermediate tasks. These intermediate tasks may be finding a square root, or typing a character on a keyboard. Such operations are often performed many times in the running of one program and may be coded as subroutines. To eliminate the need of writing the complete set of instructions each time the operation must be performed, the JMS (jump to subroutine) instruction is used. The JMS instruction stores a pointer address in the first location of the subroutine and transfers control to the second location of the subroutine. After the subroutine is executed, the pointer address identifies the next instruction to be executed. Thus, the programmer has at his disposal a simple means of exiting from the normal flow of his program to perform an intermediate task and a means of returning to the correct location upon completion of the task. (This return is accomplished using indirect addressing, which is discussed elsewhere in this chapter.)

The following example illustrates the action of the JMS instruction:

```
Location    Content
PROGRAM
   200      JMS 350     (This instruction stores 0201 in loca-
                         tion 350 and transfers program con-
                         trol to location 351.)
```



```
   201      DCA 270     (This instruction stores the contents
                         of the AC in location 270 upon return
                         from the subroutine.)
    .           .
    .           .
    .           .
SUBROUTINE
   350      0000        (This location is assumed to have an
                         initial value of 0000; after JMS 350
                         is executed, it is 0201.)
   351      i i i       (First instruction of subroutine)
    .           .
    .           .
    .           .
   375      JMP I 350   (Last instruction of subroutine)
```

The following should be kept in mind when using the JMS:

1. The value of the PC (the address of the JMS instruction +1) is always stored in the first location of the subroutine, replacing the original contents.
2. Program control is always transferred to the location designated by the operand + 1 (second location of the subroutine).
3. The normal return from a subroutine is made by using an indirect JMP to the first location of the subroutine (JMP I 350 in the above example); (Indirect addressing, as discussed in this chapter effectively transfers control to location 201).
4. When the results of the subroutine processing are contained in the AC and are to be used in the main program, they must be stored upon return from the subroutine before further calculations are performed. (In the above example, the results of the subroutine processing are stored in location 270.)

ARITHMETIC OPERATIONS

One arithmetic instruction is included in the order code, the two's complement add (TAD). Using this instruction, routines can easily be written to perform addition, subtraction, multiplication, and division in two's complement arithmetic.

Two's Complement Arithmetic

In two's complement arithmetic, addition, subtraction, multiplication, and division of binary numbers are performed in accordance with the common rules of binary arithmetic. In the PDP-8/E, as in other machines utilizing complementation techniques, negative numbers are represented as the complements of positive numbers, and subtraction is achieved by complement addition. Representation of negative values in one's complement arithmetic is slightly different from that in two's complement arithmetic.

The one's complement of a number is the complement of the absolute positive value; that is, all 1s are replaced by 0s and all 0s are replaced by 1s. The two's complement of a number is equal to one plus the one's complement of the number.