

The PDP-8 Emulator Program User's Manual

Appendix A: PDP-8 Instructions

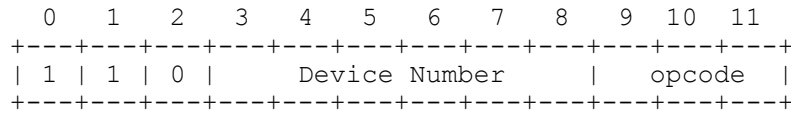
A.1 Memory Reference Instructions: Opcodes 0 - 5

0	1	2	3	4	5	6	7	8	9	10	11		
	Op-code		I		M		O	F	F	S	E	T	

Bits 0 - 2 : Operation Code
 Bit 3 : Indirect Addressing Bit (0:Direct/1:Indirect)
 Bit 4 : Memory Page (0:Zero Page/1 Current Page)
 Bits 5 - 11 : Offset Address

Assembler Mnemonic	Machine Code	Effect
AND	0nnn	logical AND $C(AC) \leftarrow C(AC) \text{ AND } C(EAddr)$
TAD	1nnn	Twos Complement Add $C(AC) \leftarrow C(AC) + C(EAddr)$ If carry out then complement Link
ISZ	2nnn	Increment and Skip on Zero $C(EAddr) \leftarrow C(EAddr) + 1$ If $C(EAddr) = 0$ then $C(PC) \leftarrow C(PC) + 1$
DCA	3nnn	Deposit and Clear Accumulator $C(EAddr) \leftarrow C(AC)$ $C(AC) \leftarrow 0$
JMS	4nnn	JuMp to Subroutine $C(EAddr) \leftarrow C(PC)$ $C(PC) \leftarrow EAddr + 1$
JMP	5nnn	JuMP $C(PC) \leftarrow EAddr$

A.2 Input Output Transfer Instructions: Opcode 6



Bits 0 - 2 : Opcode 6
Bits 3 - 8 : Device Number
Bits 9 - 11 : Extended Opcode (operation specification bits)

A.2.1 Keyboard - Device #3

Assembler Mnemonic	Machine Code	Effect
KCF	6030	Clear Keyboard Flag
KSF	6031	Skip on Keyboard Flag set
KCC	6032	Clear Keyboard flag and aCcumulator
KRS	6034	Read Keyboard buffer Static AC4..AC11 <- AC4..AC11 OR Keyboard Buffer
KRB	6036	Read Keyboard Buffer dynamic C(AC) <- 0; Keyboard Flag <- 0; AC4..AC11 <- AC4..AC11 OR Keyboard Buffer

A.2.2 Printer (CRT) - Device #4

Assembler Mnemonic	Machine Code	Effect
TFL	6040	set prinTer FLaG
TSF	6041	Skip on prinTer Flag set
TCF	6042	Clear prinTer Flag
TPC	6044	load prinTer buffer with aCcumulator and Print Printer Buffer <- AC4-11
TLS	6046	Load prinTer Sequence Printer Flag <- 0; Printer Buffer <- AC4-11

A.2.3 Interrupt System - Device #0

Assembler Mnemonic	Machine Code	Effect
SKON	6000	Skip if the interrupt system is on and turn the interrupt system off
ION	6001	Execute the next instruction then turn the interrupt system on
IOF	6002	Turn the interrupt system off

A.3 Microinstructions: Opcode 7

A.3.1 Group 1 Microinstructions (Bit 3 = 0)

0	1	2	3	4	5	6	7	8	9	10	11													
+	+	+	+	+	+	+	+	+	+	+	+													
	1		1		1		0		cla		c11		cma		cml		rar		ral		0/1		iac	
+	+	+	+	+	+	+	+	+	+	+	+													

8

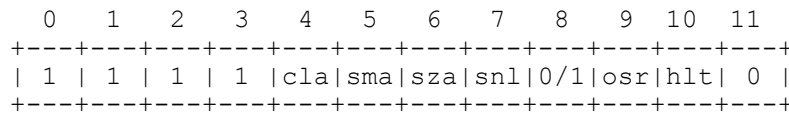
rotate 1 position if 0
2 position(s) if 1

Sequence Numbers are in ()

Assembler Mnemonic	Machine Code	Effect
NOP	7000	No Operation
CLA	7200	CLear Accumulator (1)
CLL	7100	CLear Link (1)
CMA	7040	CoMplement Accumulator (2)
CML	7020	CoMplement Link (2)
IAC	7001	Increment ACumulator (3)
RAR	7010	Rotate Accumulator and link Right (4)
RTR	7012	Rotate accumulator and link Right Twice (4)
RAL	7004	Rotate Accumulator and link Left (4)
RTL	7006	Rotate Accumulator and link left Twice (4)

Group One microinstructions may be freely combined as long as the combination makes sense. Order of operations is determined by the sequence number.

A.3.2 Group 2 Microinstructions (Bit 3 = 1, Bit 11 = 0)



|
1 in bit 8 : reverses skip sensing of bits 5,6,7

Sequence Numbers are in ()

Assembler Mnemonic	Machine Code	Effect
SMA	7500	Skip on Minus Accumulator (1)
SZA	7440	Skip on Zero Accumulator (1)
SNL	7420	Skip on Nonzero Link (1)

Combinations of SMA, SZA, and/or SNL will skip if at least one condition is true (OR Subgroup)

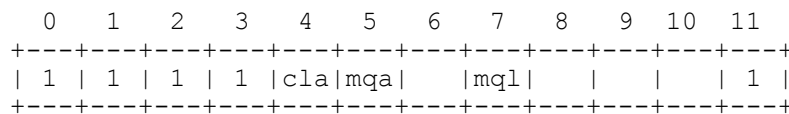
SPA	7510	Skip on Positive Accumulator (1)
SNA	7450	Skip on Nonzero Accumulator (1)
SZL	7430	Skip on Zero Link (1)

Combinations of SPA, SNA and/or SZL will skip when all conditions are true (AND Subgroup)

SKP	7410	SKiP always (1)
CLA	7600	CLear Accumulator (2)
OSR	7404	Or Switch Register with accumulator (3)
HLT	7402	HaLT (3)

Group Two microinstructions many be combined as long as instructions from the OR and the AND subgroups are not mixed and the combination makes sense. Order of operations is determined by sequence number.

A.3.3 Group 3 Microinstructions (Bit 3 = 1, Bit 11 = 1)



Sequence Numbers are in ()

Assembler Mnemonic	Machine Code	Effect
CLA	7601	CLear Accumulator (1)
MQL	7421	Load MQ register from AC and Clear AC (2)
MQA	7501	C(MQ) <- C(AC); C(AC) <- 0; Or AC with MQ register (2) C(AC) <- C(AC) Or C(MQ)
SWP	7521	SWap AC and MQ registers (3)
CAM	7621	Clear AC and MQ registers (3)

Appendix B: PDP-8 Addressing Modes

B.1 Zero Page Addressing (Bit 4 = 0)

Effective Address \leftarrow 00000 + Offset where '+' is the concatenate operation

B.2 Current Page Addressing (Bit 4 = 1)

Effective Address \leftarrow Old_PC0..Old_PC4 + Offset

where '+' is the concatenate operation. Old_PC0..Old_PC4 are bits 0 thru 4 of the PC *before* it is incremented (i.e. address of the current instruction).

B.3 Indirect Addressing (Bit 3 = 0)

If Zero Page Addressing (Bit 3 = 0) Then

Effective Address \leftarrow C(00000 + Offset)

If Current Page Addressing (Bit 3 = 0) Then

Effective Address \leftarrow C(Old_PC0-4 + Offset)

where '+' is the concatenate operation.

B.4 Autoindexing (Bit 3 = 0, Bit 4 = 0, Offset = 010o - 017o)

Addresses 0010o - 0017o are special AutoIndex Registers.

Whenever one of these locations is addressed indirectly, its contents is first incremented then used as the address of the Effective Address.

$C(\text{AutoIndex_Register}) \leftarrow C(\text{AutoIndex_Register}) + 1$

Effective Address \leftarrow C(AutoIndex_Register)

where AutoIndex_Register is an address in the range 0010o - 0017o.

This Page Left Blank

Appendix C: PDP-8 Emulator Program Commands

C.1 Main Menu

Debugger Screen
Editor
Help
Run PDP-8 Pgm Screen
Quit

C.2 Debugger Screen

Most instructions apply to the address contained in the PC.

A : Assemble instruction
B : Set/Clear Breakpoint at current value of PC
D : Deposit value to current address in PC; Advance PC. The D is optional.
G : Go or execute program. If G is followed by an integer, set breakpoint at that address
L : Load object code file into PDP-8 memory
M : Move currently displayed page; Show page containing address following 'M' command.
P : Set PC. If no numeric parameter, display page pointed to by PC
R : Initialize CPU by resetting memory and all registers to 0
S : Set Switch Register
Q : Quit Debugger and return to Main Menu
U : Unassemble instruction at current value of PC
W : Write to file contents of PDP-8 memory

(sp): Single Step; Execute instruction at current value of PC. Unassemble the instruction; Advance PC

Any input preceded by a single quote is read as input to a PDP-8 program when single-stepping.

The arrow keys move the PC.

Parameters if needed are entered on same line as command.

"A" requires an assembler language instruction.

"D", "M", "S" require a (signed) octal integer.

"L", "W" require filenames using MS-DOS filenames conventions.

"B", "P", "R", "Q", and "U" do not require parameters.

An integer parameter is optional for "G".

Illegal commands are ignored.

Ctrl/C will interrupt a PDP-8 program.

Only one breakpoint can be active; the "B" command toggles the breakpoint.

C.3 Editor

C.3.1 Editor - Menu Mode Commands

Assemble:	Assembler Source File to PDP-8 Memory. Error messages will appear in Text Area. User has option of creating a .LST file if assembly is successful.
Edit Text:	Edit Current Text
Include File:	Include File at Cursor
New Text:	Erase Current Text
Read File:	Read in File to Edit
Save File:	Save Text to File
Quit:	Quit to Main Menu

User will be prompted for file name if necessary.

C.4.2 Editor - Text Mode Commands

Arrow Keys: Move cursor

[Ins]	Toggle between Insert and Overwrite Mode
[Del]	Delete character under cursor
Ctrl/Y	Delete line
[BkSp]	Delete character in front of cursor. If cursor at front of line, merge current line with previous possible.
[Home]	Go to front of line.
[End]	Go to end of line

[PgUp]	Go up one video page
[PgDn]	Go down one video page

[Enter] Insert new line.

[Esc] or [F10] Return to Menu Mode

C.4 Run PDP-8 Screen

SwitchReg:	Enter Value into Switch Register
Deposit:	C(MB) <- C(Switch Register) C(C(CPMA)) <- C(MB) i.e. write to memory C(CPMA) <- C(CPMA) + 1; C(PC) <- C(PC) + 1 i.e. increment CPMA and PC registers
Addr Load:	C(CPMA) <- C(Switch Register)
Examine:	C(MB) <- C(C(CPMA)) i.e. read from memory C(CPMA) <- C(CPMA) + 1; C(PC) <- C(PC) + 1 i.e. increment CPMA and PC registers
Go:	Execute Program at Current Value of CPMA
Clear:	Zero out all CPU registers; clear screen
Punch:	Open/Close File for Paper Tape Punch
Reader:	Open/Close File for Paper Tape Reader
Quit:	Quit to Main Menu

Ctrl/C will Halt a PDP-8 program.

Note : C() denotes "contents of"

This Page Left Blank

Appendix D - Binary and Octal Integers

D.1 Unsigned Binary Integers

In decimal notation, there are the ten digits 0 through 9. Integer values are represented using positional notation where the position of the digit determines its weight or value as a power of ten. For example the integer 243 is 2 hundreds (10^2) plus 4 tens (10^1) plus 3 ones (10^0). In binary notation, there are only two digit, 0 and 1, called bits (short for binary digit). Positional notation is also used to represent integer values but the position of the bit determines its weight as a power of two. Thus 101011 is 1 thirty-two (2^5) plus 0 sixteens (2^4) plus 1 eight (2^3) plus 0 fours (2^2) plus 1 two (2^1) plus 1 one (2^0) or 43 decimal.

This technique of writing out any unsigned binary integer in terms of its powers of two can be used to convert any unsigned binary integer to its decimal equivalent.

Example: Convert 1011101 to decimal

$$1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 64 + 16 + 8 + 4 + 1 = 93$$

Converting decimal values to unsigned binary value can be done by subtracting out powers of two. Begin by listing the powers of two from *right to left*. Then starting with the left-most, highest power of two try to subtract each from the decimal value to be converted. If the power is too big, place a 0 beneath that power of two and go to the next power; if the power is equal or less than the decimal value, subtract it out, place a 1 under the power of two and continue with the diminished decimal value. The final string of zeros and ones will be the unsigned binary integer representation. (Showing leading zeros is acceptable)

Example: Convert 104 to binary

104	128	64	32	16	8	4	2	1	<- increasing
- 64									powers of two
----	0	1	1	0	1	0	0	0	
40	---	---	---	---	---	---	---	---	
- 32									

8									
- 8									

0	Therefore 104 decimal equals 01101000 binary								

The PDP-8 stores an unsigned binary integers in a twelve bit word. This allows a range of unsigned binary integers from 000000000000 to 111111111111 = 4095. The value of the left-most bit is 2^{11} or 2048. It should be noted that all representations of unsigned integers are twelve bits long and may need to include leading zeros.

Because it takes so many bit to represent even a moderately sized integer, documentation for the PDP-8 makes use of octal (base 8) notation. In octal notation, the digits run from 0 to 7 and the weights assigned to each position is a power of 8. For example 243o (the suffixed 'o' denotes "octal") equals 2 sixty-fours (8^2) plus 4 eights (8^1) plus 3 ones (8^0) or 163 decimal. It is also easy to convert binary to octal and back .

To convert binary to octal, Start with the binary representation, group the bits by three, and convert each group to its corresponding value between zero and seven. So 000 is 0, 001 is 1, 010 is 2 etc. (see table below).

binary	octal	binary	octal
000	0	100	4
001	1	101	5
010	2	110	6
011	3	111	7

Thus

$$101111001100 = 101\ 111\ 001\ 100 = 5714_o.$$

The same technique reversed can be used to convert an octal integer to an unsigned binary integer by substituting the corresponding three bit representation for each octal digit. So

$$2506_o = 010\ 101\ 000\ 110 = 010101000110.$$

Since the PDP-8 has a twelve bit word length, the range of unsigned octal integers is 0000_o to 7777_o. All octal representations are 4 digits and may need to include leading zeros.

D.2 Signed Binary Integers: Twos Complement Notation

Given a fixed length representation for integers (the PDP-8 uses twelve bits) signed binary integers can be represented by assigning a negative value to the left-most bit position. In the case of the PDP-8, the left-most bit position is given a value of -2^{11} .

The advantages to this technique, called two's complement representation, are many. First, converting a binary representation to its decimal equivalent by writing it as a sum of powers of two (including the leading negative value) still works. Second, methods used to add two unsigned binary integer representations will also add twos complement representations. Three, negative integers are easy to detect because their left-most bit is 1. Four, there is a very easy method for finding the twos complement representation of any negative binary integer (discussed below). Finally, we can subtract by adding a negative integer.

To obtain the two's complement representation for a negative binary integer, use the technique, called "complement and add one" (or invert and increment). Starting with the twelve bit unsigned binary representation of the integer, complement each bit (convert 0 to 1 and 1 to 0) then add one to the result.

Example: Find the twos complement representation of -216

```

000011011000  <- unsigned binary integer of 216
111100100111  <- complement
+               1  <- add 1
-----
111100101000

```

The "complement and add one" method will also convert (negate) any negative twos complement binary representation back to its positive counterpart.

Example: Undo the previous example

```

111100101000
000011010111  <- complement
+             1  <- add 1
-----
000011011000
```

Twos complement representations of signed integers are displayed in octal format by using the "group by three and convert" method to convert binary to octal. Thus $111100101000 = 111\ 100\ 101\ 000 = 7450_o$. Note that octal integers between 4000_o and 7777_o are negative integers in twos complement representation.

On the PDP-8, the range of signed integers runs from $-2048\ (4000_o)$ to $2095\ (3777_o)$. You should check that this is correct.

