MDP-80 Simulator

User Manual

By
Ronald T. Kneusel

1993

The MDP-80 simulator was designed to ease the task of learning low level machine operations and as an introduction to programming microprocessors. This manual describes the operation of the simulator only and is not a tutorial. Basic knowledge of machine code programming is assumed.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 What Is It?

The MDP-80 simulator is a Macintosh application that simulates the operation of a small microcomputer based on the MDP-8008 microprocessor (which is itself a simulation as it is not a 'real' microprocessor). This simulated computer has a total of 14k (14,336 bytes) of memory with the top 2k (from addresses F800 through FFFF) reserved for the system monitor ROM. The remaining 12k is RAM memory (addresses 0000 through 2FFF). Reading and writing to addresses in the range 3000 through F7FF will not cause an error but is the functional equivalent of reading and writing to empty sockets on the motherboard of an actual computer. The MDP-80 is connected to a simple terminal and keyboard. Input/Output is accomplished by checking specific memory locations. The microprocessor does not support interrupts and there is no form of disk I/O. The simulator provides a means for loading a program to memory from a file on the Macintosh, but programs written in the simulator cannot be saved to disk.

1.1.1 The MDP-8008 Microprocessor

A summary of the MDP-8008 microprocessor instruction set is given in Table A.1. The MDP-8008 is an 8-bit microprocessor with one accumulator and two general purpose 8-bit registers, X and Y. The microprocessor uses page 2 (0100 through 01FF) as a system stack with the current value of the stack register accessible via the X register. In addition, the MDP-8008 has a special purpose 'processor status register' (P) which contains flags that indicate the current state of the microprocessor. These flags are tested by the branch instructions. The flags are negative (N), carry (C), zero (Z), less than (L), and greater than (G). For more information about the MDP-8008 see Appendix A.

1.1.2 The System Monitor

The system monitor is itself an MDP-8008 program that will allow the user to examine and change memory and to run programs. It is run automatically at startup, displaying a message identifying the version. It then prints an '*' and waits for a line of input. The monitor is fully described in the first part of Chapter 2. The monitor also provides a collection of highly useful subroutines that will greatly ease the task of writing programs. These include input (numbers, lines of text) and output (characters, strings, and numbers) subroutines. See Appendix B for a complete description.

1.2 Features

The MDP-80 features an 'invisible' simulator that only makes its presence known when loading programs and memory from disk. The system monitor is a true MDP-8008 program and not an extension of the simulator. The source code for the system monitor is included with the simulator package in the *MDP-8008 Assembler User Manual*. Not being part of the simulator program makes the code in the system monitor available to the user, thereby making input and output much easier.

The assembler should be used for all but the smallest programs. On Macintoshes using MultiFinder or System 7.x it is easy to run the assembler, the simulator, and a text editor (not included) at the same time and thereby create a fast write/assemble/execute loop.

The simulator allows for the loading of assembly object files from disk along with the loading and saving of memory. The ability to load and save memory makes it possible to work on programs in machine code before using the assembler.

Chapter 2

Quick Start

This chapter serves as a quick introduction to the simulator and describes the system monitor commands. A sample program is also presented.

2.1 System Monitor Commands

The system monitor understands five single letter commands regardless of spacing and case. Table 2.1 summarizes these commands.

2.1.1 Examine Command

The 'X' (eXamine) command, followed by an address of one to four digits, will produce a ten line hexadecimal dump of memory beginning at the address entered. For example,

Table 2.1: System Monitor Commands.

Command	Function	Example
X 'address'	eXamine memory, dump 10 lines in hexadecimal	X 400
L 'address'	List memory, disassemble 10 lines	L 400
S 'address'	Substitute memory, enter new value, '!' to end	S 400
G 'address'	Go (run program)	G 400
'address'	display contents of the address entered	12EF
!	clear the screen and home the cursor	!

```
*X 400  (entered by the user)
0400-A9 41 20 F8 76 60 00 00
0408-00 00 00 00 00 00 00 00
0410-00 00 00 00 00 00 00 00
0418-00 00 00 00 00 00 00 00
0420-00 00 00 00 00 00 00 00
0428-00 00 00 00 00 00 00 00
0430-00 00 00 00 00 00 00 00
0438-00 00 00 00 00 00 00 00
*
```

2.1.2 List Command

For a disassembled listing of memory use the 'L' (List) command. For example,

```
*L 400   (again entered by the user)
0400- A9 41    LDA #41
0402- 20 F8 76 JSR $F876
0405- 60       RTS
0408- 00       BRK
0409- 00       BRK
040A- 00       BRK
040B- 00       BRK
040C- 00       BRK
040D- 00       BRK
040E- 00       BRK
040F- 00       BRK
0410- 00       BRK
0411- 00       BRK
0412- 00       BRK
0413- 00       BRK
0414- 00       BRK
*
```

2.1.3 Substitute Command

To change the contents of memory use the 'S' (Substitute) command followed by an address. To change the value of a memory location enter the new value, to leave it unchanged press 'return'. To exit substitute mode enter '!'. For example,

```
*S 400   (user input)
0400-00  A9    (entered by the user)
0401-00  41
0402-00  20
0403-00  F8
0404-00  76
0405-00  60
0406-00  !     (end input)
```

```
*
```

2.1.4 Go Command

Run a machine language program by entering 'G' followed by the first address in the program. Be sure to end all programs with a `60 (RTS)` instruction to return to the monitor. For example, using the code entered above,

```
*G 400         (user input)
A*             (program outputs 'A' then back to monitor)
```

2.1.5 Other Commands

Typing '!' will clear the screen and home the cursor. Typing a valid hexadecimal address (one to four digits) will display the current contents of that address.

2.2 Example - the Alphabet

As an illustration of the simulator we now present a small example program to print the letters of the alphabet. This program uses the `_COUT` monitor subroutine at location `F876` to print the character in the accumulator. Try this example for yourself by entering the values listed,

```
MDP-80 Monitor ROM version 1.0

*S 400
0400-00  A9
0401-00  41
0402-00  20
0403-00  F8
0404-00  76
0405-00  AE
0406-00  88
0407-00  5B
0408-00  D1
0409-00  04
040A-00  02
040B-00  A9
040C-00  0D
040D-00  20
040F-00  F8
0410-00  76
0411-00  60
0412-00  !
```

```
*G 400
ABCDEFGHIJKLMNOPQRSTUVWXYZ
*
```

As an exercise, change the program to output the alphabet in lowercase (ASCII values `61` through `7A`) by changing `401` to `61` and `407` to `7B`.

Chapter 3

The Simulator Application

The simulator application uses two menus, *File* and *Edit*. The *Edit* menu is for desk accessories and is not used.

3.1 The File Menu

The *File* menu options are *Load file*, *Load memory*, *Save memory*, and *Quit*. The last one is self-evident. The *Load file* option loads assembly language object files into the simulator's memory. It brings up a standard Macintosh dialog box for the user to enter the name of an assembler object file (.OBJ) to load. The simulator displays the beginning address of the file and returns to the system monitor.

The *Load memory* and *Save memory* options read or write the entire contents of the simulator's memory to a disk file. This is useful when writing programs without using the assembler. To use this feature make a copy of the supplied file *MDP-80 memory* in the Finder and name it what you will. Then, when in the simulator and ready to save the memory, select *Save memory* and use the dialog box to select the file just created. The entire contents of the simulator's memory will be written to this file. Use *Load memory* to restore a memory image. If a disk error occurs the simulator will indicate the error code and exit.

3.2 Breaking an Infinite Loop

The mouse button resets the simulator. Pressing it will interrupt the program currently running and return to the monitor. Embedding a BRK (`00`) instruction in a program will cause the simulator to jump to the address `FFFA` which contains a jump to the break routine.

Appendix A

MDP-8008 Reference

The MDP-8008 microprocessor has 52 instructions and 7 addressing modes. Table A.1 gives a summary of the MDP-8008 instruction set. The instruction names are given in Table‘A.2. The second column of Table A.1 indicates with an ‘*’ which of the processor status flags are affected by each instruction. The flags are (N) negative (bit seven is set), (C) carry, (Z) zero, (L) less than, and (G) greater than.

A.1 MDP-8008 addressing modes

An instruction's addressing mode determines the way in which the instruction will access and use memory. The seven modes supported by the MDP-8008 are implied, immediate, absolute, absolute indexed, zero page, relative, and indirect indexed.

Implied addressing

The implied address in implied addressing is the accumulator. Implied mode is used by single byte instructions that only act on the accumulator and do not reference memory.

Immediate addressing

The value used in immediate mode addressing is the next byte of the program, or the first operand. The target of the instruction is one of the microprocessor's registers which is loaded with the immediate value. For example, `A9 F3` will load the accumulator with the value `F3`.

Absolute addressing

An 8-bit microprocessor like the MDP-8008 can access a total of 65536 memory locations. It therefore takes two bytes to represent the address of any memory location. Absolute addressing uses two operand bytes (the first two bytes of the program following the byte containing the opcode) to represent the address of a memory location. For example, `B9 1E 37` will store the current value of the accumulator in location `1E37`.

Absolute indexed addressing

Similar to absolute addressing, absolute indexed addressing uses as a target address the value obtained when the address represented by the operand bytes is added to the current value of the X register. If the X register contains `14` and the instruction is `BD 1E 38` the contents of the accumulator will be stored in location `1E38` + `14` = `1E4C`. Absolute indexed addressing is particularly useful when dealing with lists.

Indirect indexed addressing

The target address in indirect index addressing is found by taking the absolute address in the operands, reading the two byte address that starts at the given address and adding the contents of the X register to get the final address. For example, suppose the instruction being executed (in assembly form) is `61 2E 33`. If the values stored in memory locations `2E33` and `2E34` are `14` and `1C` and the current value of the X register is `A4` then the accumulator will be loaded with the contents of location `141C` + `A4` = `14C0`.

Zero page addressing

Zero page addressing takes advantage of the fact that the first 256 memory locations can be addressed in a single byte (`00` to `FF`). Therefore, all instructions using the zero page are only two bytes long, which can add up to a significant savings if frequently used locations are in the zero page.

Relative addressing

Only the branch instructions use relative addressing. Instead of specifying an absolute location in memory, a relative address is a positive or negative offset from the current address. The use of relative addressing allows one to write code that will run at any location in memory. Relative addressing uses one operand and as a result can reference locations that are -128 to +127 bytes from the current location. Tables A.3 and A.4 can be used to determine the appropriate operand for a specific branch.

Table A.1: Summary of MDP-8008 instruction set.

Mnemonic	N-C-ZLG-	Implied	Immediate	Absolute	Abs,X	Zero	Relative	Indirect
ADC	*-*-*---		A5	B5	BE			
AND	----*---		A0	99				
BCC	--------			D5			D4	
BCS	--------			DB			DA	
BEQ	--------			D3			D2	
BGT	--------			D9			D8	
BLT	--------			D7			D6	
BNE	--------			D1			D0	
BNG	--------			DD			DC	
BPL	--------			DF			DE	
BRA	--------						E0	
BRK	--------	00						
CLC	--*-----	54						
CMP	----***-		88	89	8A	CB		
CPX	----***-		86	85		CC		
CPY	----***-		8B	8C		CD		
DCA	----*---	AF						
DEC	----*---			45	34			
DEX	----*---	C7						
DEY	----*---	C5						
EOR	----*---		A2	97				
INA	----*---	AE						
INC	----*---			46	35			
INX	----*---	E8						
INY	----*---	EE						
JMP	--------			4C				
JSR	--------			20				
LDA	*---*---		A9	AA	AD	C0		61
LDX	*---*---		A8	AB		C2		
LDY	*---*---		A6	AC		C4		
LSH	--*-----	26						
NOP	--------	EA						
ORA	----*---		A1	98				
PHA	--------	3C						
PHP	--------	33						
PLA	--------	3D						
PLP	--------	36						
ROL	--*-----	28						
ROR	--*-----	27						
RSH	--*-----	29						
RTS	--------	60						
SBC	*-*-*---		A4	BB	BA			
SEC	--*-----	55						
STA	--------			B9	BD	C6		63
STX	--------			B8		C8		
STY	--------			B6		CA		
TAX	--------	14						
TAY	--------	13						
TSX	--------	07						
TXA	--------	12						
TXS	--------	10						
TYA	--------	11						

Table A.2: MDP-8008 instruction names.

Mnemonic	Function
ADC	Add to accumulator with carry
AND	AND accumulator with memory
BCC	Branch on carry clear
BCS	Branch on carry set
BEQ	Branch on equal or zero
BGT	Branch on greater than
BLT	Branch on less than
BNE	Branch on not equal or not zero
BNG	Branch on negative set
BPL	Branch on positive (not negative)
BRA	Unconditional relative branch
BRK	Break
CLC	Clear carry flag
CMP	Compare accumulator to memory
CPX	Compare X register to memory
CPY	Compare Y register to memory
DCA	Decrement accumulator by one
DEC	Decrement memory by one
DEX	Decrement X register by one
DEY	Decrement Y register by one
EOR	Exlusive-OR accumulator with memory
INA	Increment accumulator by one
INC	Increment memory by one
INX	Increment X register by one
INY	Increment Y register by one
JMP	Jump to address
JSR	Jump to subroutine at address
LDA	Load the accumulator with memory
LDX	Load the X register with memory
LDY	Load the Y register with memory
LSH	Bit shift accumulator left, no carry
NOP	No operation
ORA	OR accumulator with memory
PHA	Push accumulator on stack
PHP	Push processor status on stack
PLA	Pull accumulator from stack
PLP	Pull processor status from stack
ROL	Rotate accumulator to the left with carry
ROR	Rotate accumulator to the right with carry
RSH	Bit shift accumulator right, no carry
RTS	Return from subroutine
SBC	Subtract memory from accumulator with carry
SEC	Set carry flag
STA	Store accumulator in memory at address
STX	Store X register in memory at address
STY	Store Y register in memory at address
TAX	Transfer accumulator to X register
TAY	Transfer accumulator to Y register
TSX	Transfer stack pointer to X register
TXA	Transfer X register to accumulator
TXS	Transfer X register to stack pointer
TYA	Transfer Y register to accumulator

Table A.3: Forward relative branch table. Find the number of bytes forward by locating the value in the body and reading the most significant and least significant hexadecimal digits from the row and column headings respectively. Example, to branch forward 73 bytes, locate 73 in the body of the table and choose 4 from the row and 9 from the column to make 49.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

Table A.4: Backward relative branch table. Find the number of bytes backwards by locating the value in the body and reading the most significant and least significant hexadecimal digits from the row and column headings respectively. Example, to branch backwards 73 bytes, locate 73 in the body of the table and choose B from the row and 7 from the column to make B7.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	128	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113
9	112	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97
A	96	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81
B	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65
C	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49
D	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
E	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
F	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Table A.5: Page 2 special locations.

Location	Name	Purpose
0200	KOUT	Put character to print here
0201	CSTB	And set this to non-zero to print
0202	KBDS	Set to non-zero to wait for a keypress
0203	KEYB	Get character typed from here

A.2 MDP-80 System Memory Map and ASCII Table

Figure A.1 shows how memory is used by the MDP-80 simulator. Zero page is available to the user if locations below `15` are unaffected. All of page one (0100 - 01FF) is reserved for the microprocessor's system stack. Several locations in page two (0200 - 02FF) are reserved for keyboard and screen I/O as shown in Table A.5. While it is possible to use these locations directly there exist system monitor routines to handle keyboard and terminal I/O. Table A.6 summarizes the 7-bit ASCII code.

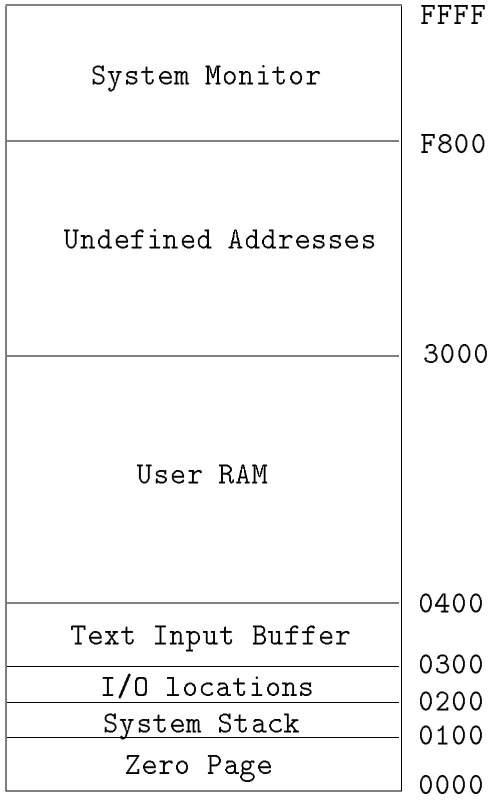

Figure A.1: MDP-80 memory map.

Table A.6: 7-bit ASCII codes.

Char	Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char	Hex	Dec
ctl-@ (NUL)	00	00	space	20	32	@	40	64	sp	60	96
ctl-A (SOH)	01	01	!	21	33	A	41	65	a	61	97
ctl-B (STX)	02	02	"	22	34	B	42	66	b	62	98
ctl-C (ETX)	03	03	#	23	35	C	43	67	c	63	99
ctl-D (EOT)	04	04	$	24	36	D	44	68	d	64	100
ctl-E (ENQ)	05	05	%	25	37	E	45	69	e	65	101
ctl-F (ACK)	06	06	&	26	38	F	46	70	f	66	102
ctl-G (BEL)	07	07	’	27	39	G	47	71	g	67	103
ctl-H (BS)	08	08	(	28	40	H	48	72	h	68	104
ctl-I (HT)	09	09	)	29	41	I	49	73	i	69	105
ctl-J (LF)	0A	10	*	2A	42	J	4A	74	j	6A	106
ctl-K (VT)	0B	11	+	2B	43	K	4B	75	k	6B	107
ctl-L (FF)	0C	12	,	2C	44	L	4C	76	l	6C	108
ctl-M (CR)	0D	13	-	2D	45	M	4D	77	m	6D	109
ctl-N (SO)	0E	14	.	2E	46	N	4E	78	n	6E	110
ctl-O (SI)	0F	15	/	2F	47	O	4F	79	o	6F	111
ctl-P (DLE)	10	16	0	30	48	P	50	80	p	70	112
ctl-Q (DC1)	11	17	1	31	49	Q	51	81	q	71	113
ctl-R (DC2)	12	18	2	32	50	R	52	82	r	72	114
ctl-S (DC3)	13	19	3	33	51	S	53	83	s	73	115
ctl-T (DC4)	14	20	4	34	52	T	54	84	t	74	116
ctl-U (NAK)	15	21	5	35	53	U	55	85	u	75	117
ctl-V (SYN)	16	22	6	36	54	V	56	86	v	76	118
ctl-W (ETB)	17	23	7	37	55	W	57	87	w	77	119
ctl-X (CAN)	18	24	8	38	56	X	58	88	x	78	120
ctl-Y (EM)	19	25	9	39	57	Y	59	89	y	79	121
ctl-Z (SUB)	1A	26	:	3A	58	Z	5A	90	z	7A	122
ctl-[(ESC)	1B	27	;	3B	59	[	5B	91	{	7B	123
ctl-\ (FS)	1C	28	<	3C	60	\	5C	92	|	7C	124
ctl-] (GS)	1D	29	=	3D	61	]	5D	93	}	7D	125
ctl-↑ (RS)	1E	30	>	3E	62	↑	5E	94	˜	7E	126
ctl-← (VS)	1F	31	?	3F	63	←	5F	95	DEL	7F	127

Appendix B

Useful Monitor Subroutines

The simulator's built in system monitor program occupies the 2K of memory from F800 to FFFF and contains the routines necessary to make it possible to enter, execute, and modify machine language programs. It also contains a number of useful routines for printing strings and numbers, entering data, and converting strings to numbers. Table B.1 summarizes the most useful monitor subroutines.

_COUT	F876	Output character in A

Description: Displays the character whose ASCII code is stored in the accumulator.

Registers altered: None.

_KEYIN	F87D	Input character from keyboard

Description: Waits for a key to be pressed and places the ASCII value in the accumulator.

Registers altered: A.

Table B.1: Some useful system monitor subroutines.

Name	Location	Input	Output	Use
_COUT	F876	character in A	none	Prints a character
_KEYIN	F87D	none	character in A	Read keyboard
_INPUT	F884	none	text starts 0300, X length	Input line, prompt
_INPUT1	F889	none	text starts 0300, X length	Input line, no prompt
_PRINT	F8D2	AX holds string start	none	Print a string
_PRINTHEX	F8E9	number in A	none	Print A as a number
_PRINTWORD	F920	number in AX	none	Print AX as 16-bit number
_UPPER	F928	AX holds start	none	Convert string to uppercase
_KILLB	F952	AX holds start	none	Remove blanks from a line
_LENGTH	F989	06 & 07 hold start	X length	Length of string
_NUMBER	F9DD	AX holds start	01 & 02	String to number (16-bit)
_REG	FF7F	none	none	Display current register values
_BLANKS	FFD8	number in X	none	Output X blanks

_INPUT	F87D	Input a line of text, with prompt
_INPUT1	F889	Input a line of text, no prompt

Description: Inputs a line of text. Characters are stored starting at location 0300. Backspace and delete keys work as normal. X holds the length on exit.

Registers altered: A, X.

_PRINT	F8D2	Print a null terminated string

Description: Displays the null terminated string the address of which is in A (MSB) and X (LSB).

Registers altered: A, X.

_PRINTHEX	F8E9	Print A as a hexadecimal number

Description: Display the accumulator as a two digit hexadecimal number.

Registers altered: A.

_PRINTWORD	F920	Print AX as a four digit hexadecimal number

Description: Displays AX as a four digit number. A holds the high part, X holds the low.

Registers altered: A, X.

_UPPER	F928	Convert a string to uppercase

Description: Converts the null terminated string starting at the address in AX to uppercase.

Registers altered: A, X.

_KILLB	F952	Remove the blanks from a string

Description: Removes the blanks from the null terminated string that starts at the address in AX.

Registers altered: A, X.

_LENGTH	F989	Find the length of a string

Description: Returns in X the length of the null terminated string pointed to by 06 and 07.

Registers altered: A, X.

_NUMBER	F9DD	Convert a string to a number

Description: Converts the null terminated string in AX into a 16-bit hexadecimal number stored in 01 and 02.

Registers altered: A, X, and Y.

_REG	FF7F	Display current register values

Description: Displays the current register values.

Registers altered: None.

_BLANKS	FFD8	Output blank spaces

Description: Displays blank spaces, count in X.

Registers altered: A, X.