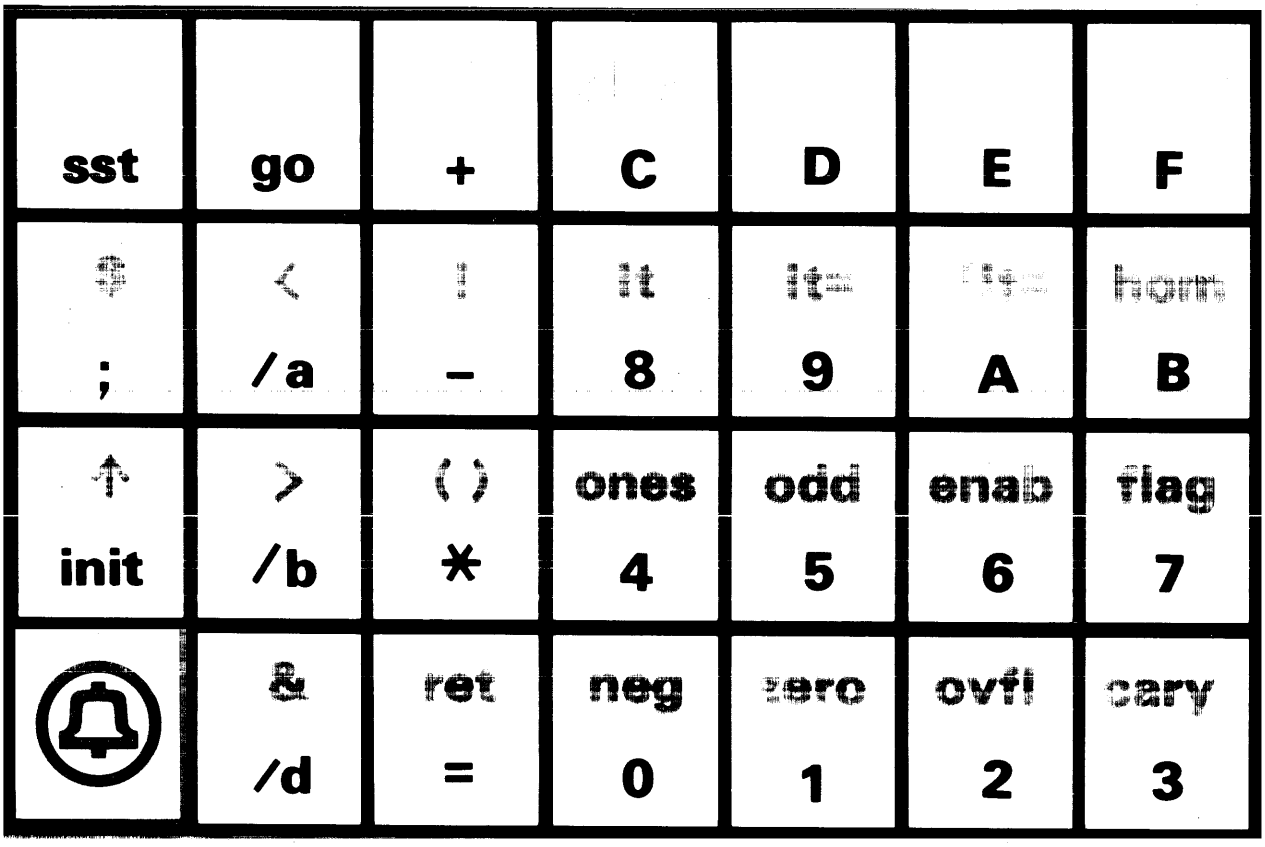
MACTUTOR



Self-Training Manual

PA-800516
Issue 1, April 1979

Prepared for the Microprocessor System Department
and the
Continuing Education Department
by the
Technical Documentation Department,
Bell Laboratories, 1978

Second Printing 1979

PA-800516
Issue 1, April 1979
AT&T Co Provisional

CONTENTS

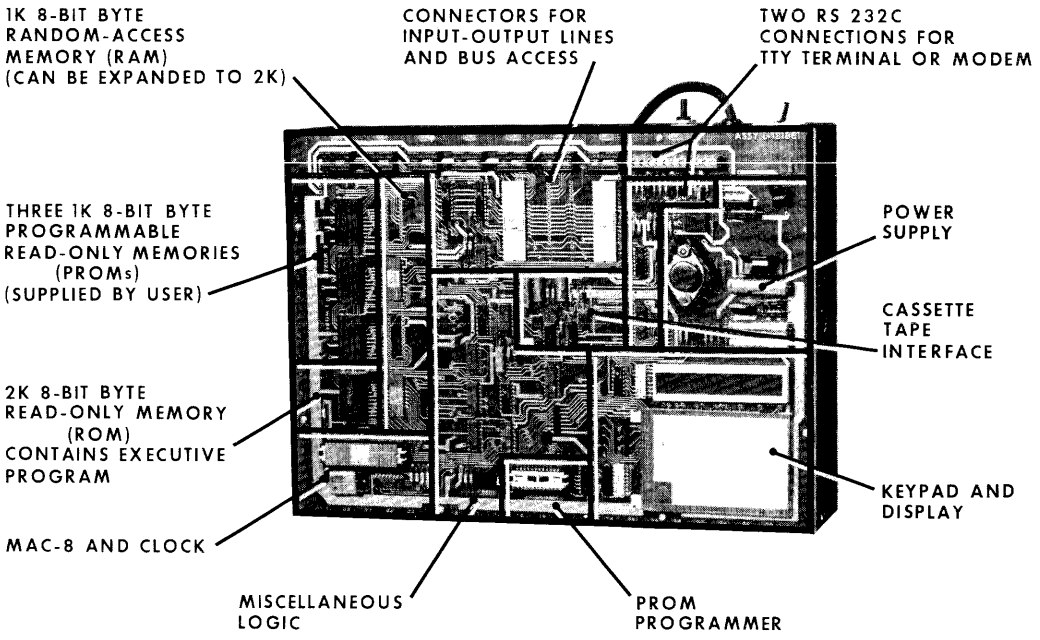
The MACTUTOR *Self-Training Manual* incorporates the introductory booklet MACTUTOR with additional programming exercises, including input/output and teletype exercises.

PA-800516 Issue 1, April 1979

DESCRIPTION

The MAC Tutor is a self-teaching machine designed to introduce you to the scope and capabilities of the MAC-8 microprocessor. The MAC Tutor allows you to:

- Enter and examine program instructions and data in memory.
- Execute the instructions.
- Examine the results.



MAC Tutor

FEATURES

- A MAC-8 microprocessor with access to 65,536 8-bit bytes of memory.

- **Read-only memory (ROM).** There are 2048 8-bit bytes of ROM that contain the resident executive program, which allows you to store, execute, and debug your programs. The executive program also contains the interface routines for peripheral devices. This program is located at hexadecimal (hex.) addresses 0000 through 07FF.

- **Random-access memory (RAM).** There are 1024 8-bit bytes of RAM that contain the registers where programs are temporarily stored, manipulated, and executed. The RAM is located at hex. addresses 1800 through 1BFF, and there are sockets for an additional 1024 bytes of RAM with hex. addresses 1400 through 17FF. You should avoid using RAM addresses above 1BB0 in your programs because the resident executive program reserves most of the locations above this address for its own use.

- **Programmable read-only memory (PROM).** There are sockets for three PROMs constituting 3072 locations with hex. addresses 0800 through 13FF. The PROMs are used for permanent storage of your programs.

- **A PROM programming socket** that is used to permanently burn in programs in PROMs.

- **An audio cassette tape interface** for storing and retrieving data at a 166-baud rate.

- **Two EIA standard connections** for a teletype terminal or a modem with a 0- through 2400-baud rate.

- **Address and data bus 16-pin connectors** for the addition of extra memory or peripheral devices other than an audio cassette tape, teletype terminal, or modem.

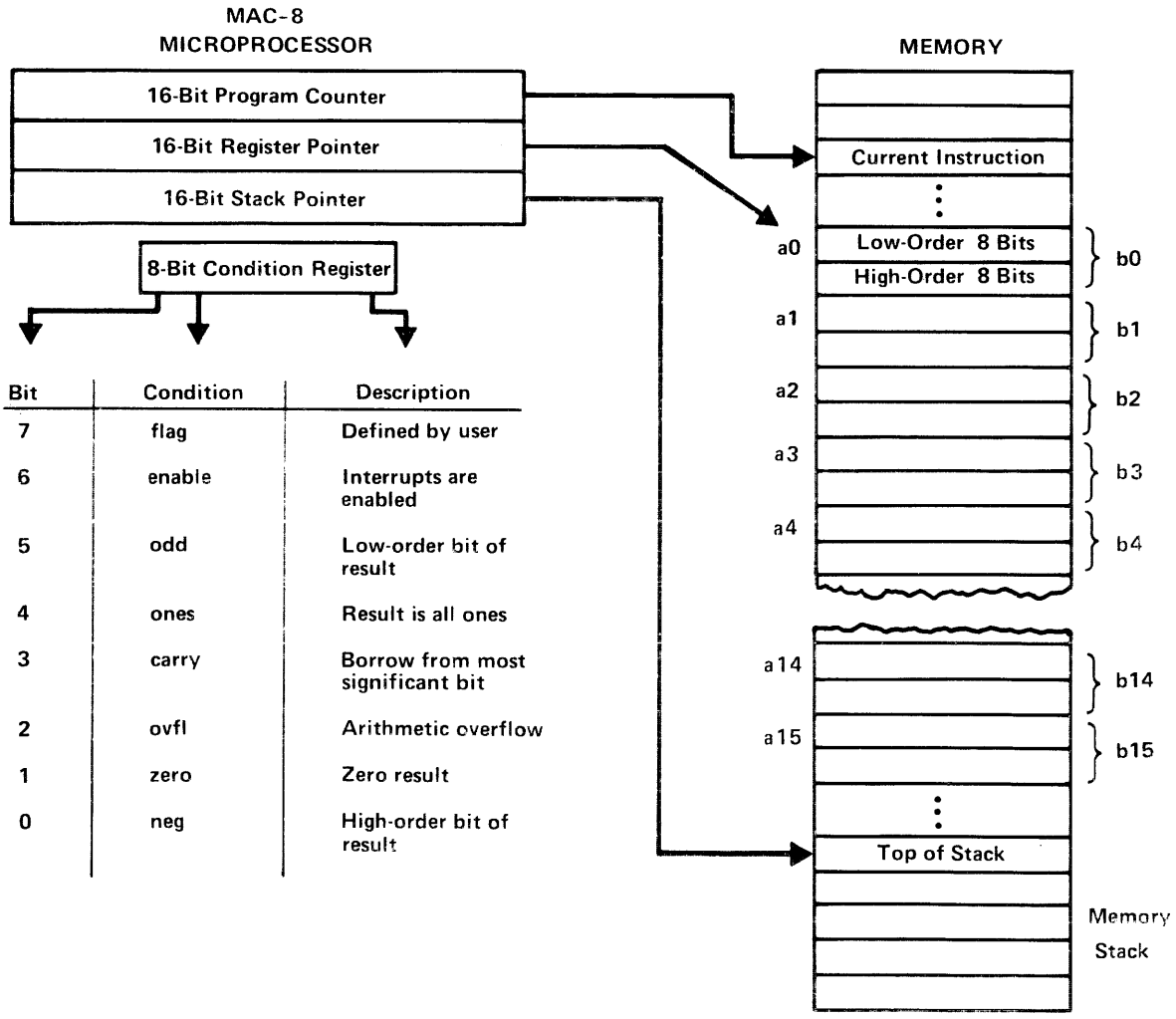
- **An on-board power supply** that allows you to operate anywhere from a 117-volt 60-Hz outlet.

- **A light-emitting diode (LED) display** that consists of eight 7-segment LED arrays, and each LED array displays one hex. digit. Because the LED arrays comprise only seven segments and to avoid confusion, the hex. digits B, C, and D appear in lower-case form, that is, b, c, and d. The hex. digits A, E, and F appear in upper-case form. Also, the hex. digit 6 has the top horizontal segment lit to distinguish it from the hex. digit b. In the following exercises, the leftmost four hex. digits of the display indicate an address in memory. The rightmost two hex. digits indicate the contents of that address (also referred to as the low contents). The remaining two hex. digits indicate the contents of the next address (referred to as the high contents).

- **A 28-button keypad.** The program instructions and data are entered via the keypad. Most of the keys have two designations, one in yellow and one in blue. You will be using only the blue designations; the yellow designations are reserved for future expansion. The keys and their functions are as follows.

KEYS			FUNCTION
0	THROUGH	9	Hexadecimal digits representing digits 0 through 15.
A	AND THROUGH	F	
init			Initialize memory.
*			Specify memory address.
=			Change memory contents.
+			Increment memory address.
-			Decrement memory address.
/a			Display register a.
/b			Display register b.
/d			Display register pointer.
go			Execute program.
sst			Execute program in single-step mode.
;			Reserved for future use.
			TTY interface.

- **Registers.** As mentioned previously, the RAM contains the registers where programs are temporarily stored, manipulated, and executed. A special register set, as shown below, comprises 32 8-bit memory locations that are grouped in pairs. The pairs provide 16 16-bit **b** registers, b0 through b15, and the low-order 8 bits of the register pairs provide 16 8-bit **a** registers, a0 through a15. In addition, there are four important registers that are located in the MAC-8 microprocessor. The register pointer (rp) is a 16-bit register that contains the lowest memory address occupied by the **a** and **b** registers. The program counter (pc) is a 16-bit register that contains the address of the program instruction that is currently being executed. The stack pointer (sp) is a 16-bit register that contains an address in a block of memory called the memory stack; a primary function of the stack is to store the address of the next instruction in the main program to be executed after a return instruction is executed in a subroutine. The condition register (cr) is an 8-bit register that contains the condition bits, which are logical indicators that can be tested for decision making.



Bit	Condition	Description
7	flag	Defined by user
6	enable	Interrupts are enabled
5	odd	Low-order bit of result
4	ones	Result is all ones
3	carry	Borrow from most significant bit
2	ovfl	Arithmetic overflow
1	zero	Zero result
0	neg	High-order bit of result

Register Set

SIMPLE OBSERVATIONS

If there is a video cassette accompanying this manual and a playback facility is available, plug the cassette in now to review the description of the MAC Tutor and to perform the following simple exercises.

Plug in the power cord and turn on the switch at the rear of the MAC Tutor. (Because the high and low contents contain random data, the data shown in the following illustrations may differ from the data shown on your MAC Tutor. Therefore, you will have to substitute the data shown on your MAC Tutor for the data given below.)

Press **init** and observe

1 8 0 0 3 d 9 0

1800 is the hex. address to which the executive program initializes memory, the digits 3 D (high contents) represent random data stored at hex. address 1801, and the digits 9 0 (low contents) represent random data stored at hex. address 1800.

Press + and observe

1 8 0 1 0 8 3 d

The memory address is incremented by 1. Note that the random data that appeared in the high contents have been shifted to the right into the low contents and the random data that are stored at hex. address 1802 appear in the high contents.

Press + and observe

1 8 0 2 0 d 0 8

The memory address is incremented again. The random data that are stored at hex. address 1802 are shifted into the low contents and the random data that are stored at hex. address 1803 appear in the high contents.

Press – and observe

1 8 0 1 0 8 3 d

The memory address is decremented; therefore, the random data that are stored at hex. address 1802 shift to the left into the high contents and the random data that are stored at hex. address 1801 reappear in the low contents.

Press * and key in 1 8 6 3 and observe

1 8 6 3 9 A 8 A

A particular memory address, 1863, is specified, but the contents are still random.

SIMPLE OPERATIONS

Press init and observe

1 8 0 0 3 d 9 0

Press = and key in A B and observe

1 8 0 0 3 d A b

The digits AB are stored at hex. address 1800.

Press + and key in 0 5 and observe

1 8 0 1 0 8 0 5

The memory address is incremented and the digits 0 5 are stored there (note that the leading 0 has to be entered). If you make an error when you enter data at an address, simply reenter all the digits that make up the data.

Press _ and observe

1 8 0 0 0 5 A b

The preceding data are stored in sequence.

When storing data at sequential addresses, press = once at the beginning. By pressing + before each entry, the MAC Tutor remains in the change memory contents, or store, mode. When storing data at nonsequential addresses, press * and enter the address, then press = and enter the data.

INTRODUCTORY PROGRAM EXERCISES

At this point, you are ready to enter and execute programs. The following program exercises illustrate the extent of the instructions understood by the MAC Tutor. The instructions for

these exercises are written in MAC-8 assembly language format and they must be converted to hex. code before they can be entered into the MAC Tutor. The hex. code is supplied for all the program exercises contained in this manual; however, the 0x prefix required by the MAC-8 assembler for hex. number representation has been omitted. To obtain maximum benefit from the program exercises, you should carefully study the comments associated with each exercise to become acquainted with the operations performed by each instruction and use the *MAC-8 Hexadecimal Coding Chart* for verification of each instruction code.

Exercise 1 — Moving Data From One Address to Another

This program exercise moves data stored at hex. address 18A0 to hex. address 18A1.

Instruction	Hex. Code	Instruction Address	Comments
b0=18A0;	C0 0F A0 18	1800 through 1803	Load the 16-bit register b0 with 16-bit hex. address 18A0. The assembly language for this instruction is bd=W, where bd is the 16-bit b register that is being used as the destination and W is the 16-bit address that is being loaded. To verify the hex. code for this instruction, look at Part 1 of the *MAC-8 Hexadecimal Coding Chart*. Under the Addressing Mode column, find the entry 16-Bit Register and 16-Bit Immediate. Under the Move column, find the entry bd = W C0 dF W(LO) W(HI) C0 is an operation code (opcode), d is the destination (register b0), F is an opcode, and W(LO) and W(HI) are the low-order eight bits (A0) and the high-order eight bits (18), respectively, of the 16-bit address.

Exercise 1 (Continued)

Instruction	Hex. Code	Instruction Address	Comments
a1=*b0;	85 10	1804,1805	Move the contents of the address pointed to by 16-bit register b0 (18A0) to 8-bit register a1. The assembly language for this instruction is ad = *bs, where ad is the a register that is being used as the destination,* indicates the contents of the address that is being pointed to, and bs is the b register that is being used as the source. In Part 1, under the Addressing Mode column, find the entry Register and Indirect and under the Move column find the entry ad = * bs 85 ds 85 is an opcode, d is the destination (register a1), and s is the source (register b0).
++b0;	68 00	1806, 1807	Increment the contents (address 18A0) of 16-bit register b0 by 1 (18A1). The assembly language is ++bd, where ++ indicates to increment and bd is the b register that is being used as the destination. In Part 2, under the Addressing Mode column, find the entry 16-Bit Register and under the Increment column find the entry ++ bd 68 d0 68 is an opcode, d is the destination (register b0), and 0 is an opcode.

Exercise 1 (Continued)

Instruction	Hex. Code	Instruction Address	Comments
*b0=a1;	81 01	1808, 1809	Move the contents of register a1 to the address specified by register b0 (18A1). The assembly language is *bd = as, where * indicates the contents of the address that is being pointed to, bd is the b register that is being used as the destination, and as is the a register that is being used as the source. In Part 1, under the Addressing Mode column, find the entry Indirect and Register and under the Move column, find the entry *bd = as 81 ds 81 is the opcode, d is the destination (register b0), and s is the source (register a1).
return;	66	180A	Return to the executive program. The opcode for this instruction is found in Part 4, under the Return Instructions and Unconditional columns.

Enter the program by keying in the commands and hex. code as follows.

init = C 0 + 0 F + A 0 + 1 8 + 8 5
+ 1 0 + 6 8 + 0 0 + 8 1 + 0 1 + 6 6

You can examine the program for errors by pressing **init** and then pressing + repeatedly. This enables you to observe that the sequence of hex. addresses and hex. code is as indicated in the previous program listing. If you find an error in the hex. code, press = and enter the correct hex. code.

Now, store arbitrary data at hex. addresses 18A0 and 18A1. For example, key in *** 1 8 A 0 = 2 3 + 7 C** — and observe

1 8 A 0 7 c 2 3

Press **init** to initialize the program counter to hex. address 1800, which is the address of the first instruction in the program, and then press **go**, which executes the program.

Exercise 1 (Continued)

Key in * **1 8 A 0** and observe

1 8 A 0 2 3 2 3

Data stored at hex. address 18A0 are now also stored at hex. address 18A1. During execution, the program uses registers a1 and b0. To examine the contents of register a1, press /a and key in **1** and observe

A 1 2 3

Register a1 contains the data to be moved.

To examine the contents of register b0, press /b and observe

b 0 1 8 A 1

Register b0 contains the new address of the data. The a and b registers can be observed individually, as above, or in sequence by pressing /a or /b and then pressing + repeatedly.

You can also observe the execution of the program by performing it in the single step mode. For example, re-store the arbitrary data 23 and 7C at hex. addresses 18A0 and 18A1, respectively. Key in * **1 8 A 0** and observe

1 8 A 0 2 3 2 3

This is the result of the execution of the program. The address 18A0 already contains 23, so key in + = **7 C** – and observe

1 8 A 0 7 c 2 3

Now, press **init** and observe

1 8 0 0 0 F c 0

If you check the program listing, you will see that this is the start (the first two bytes) of the first instruction in the program. Instead of pressing **go**, quickly press and release **sst** and observe

1 8 0 4 1 0 8 5

This is the start of the second instruction. Only the first instruction has been executed. You can verify this by checking the contents of register b0. As the Comments column indicates, register b0 should contain the hex. address 18A0. Press /b and observe

b 0 1 8 A 0

Exercise 1 (Continued)

Press * and observe

1 8 0 4 1 0 8 5

You have returned to the start of the second instruction in the program. Quickly press and release sst again and observe

1 8 0 6 0 0 6 8

which is the start of the third instruction. Verify that the second instruction has been executed by checking the contents of register a1. Register a1 should contain the contents of the address pointed to by register b0. Press /a, key in 1, and observe

A 1 2 3

Press * and sst again and observe

1 8 0 8 0 1 8 1

This is the start of the fourth instruction, and execution of the third instruction can be verified by checking the contents of register b0 again. The address in register b0 should be incremented by 1. Press /b and observe

b 0 1 8 A 1

Press * and sst and observe

1 8 0 A x x 6 6 (xx are random contents)

This is the start of the final instruction of the program. At this point, you can verify the data transfer by checking the contents of hex. address 18A0. Press*, key in 1 8 A 0, and observe

1 8 A 0 2 3 2 3

To execute the final instruction, the return to the executive program, press * and sst and observe

0 0 3 5 0 F 4 d

Hex. address 0035 is the location in the executive program that serves as a point of entry from the user program.

In the following program exercises, the instructions are entered in a manner similar to that described at the beginning of this exercise, and the programs can be executed at full speed or in the single step mode.

Exercise 2 — Loading 8-Bit Data Into Memory

This program exercise loads 8-bit data into specified memory locations using various addressing techniques.

Instruction	Hex. Code	Instruction Address	Comments
rp=1B00;	4D 0F 00 1B	1800 through 1803	Load rp with hex. address 1B00.
b0=1A00;	C0 0F 00 1A	1804 through 1807	Load register b0 with hex. address 1A00. 1A is loaded into 1B01 and 00 is loaded into 1B00 because the rp is initialized to 1B00 and low-order bits are stored first.
b1=1A00;	C0 1F 00 1A	1808 through 180B	Load register b1 with hex. address 1A00. 1A is loaded into 1B03 and 00 is loaded into 1B02.
a1=50;	80 1F 50	180C through 180E	Load register a1 (hex. address 1B02) with 50. This overwrites the 00 that was stored in the previous instruction; therefore, register b1 now stores hex. address 1A50.
*1B04=E5;	81 FF 04 1B E5	180F through 1813	Load hex. address 1B04 (register a2) with E5.
*b0=45;	81 0F 45	1814 through 1816	Load hex. address specified by register b0 (1A00) with 45.
*(b0+1)=88;	82 0F 01 88	1817 through 181A	Increment hex. address specified by register b0 (1A01) and load 88 into it.
*b1=79;	81 1F 79	181B through 181D	Load 79 into hex. address specified by register b1 (1A50).
*(b1−1)=2F;	82 1F FF 2F	181E through 1821	Decrement hex. address specified by register b1 (1A4F) and load 2F into it.
	FF	1822	Illegal opcode indicating stop.

Exercise 2 (Continued)

Execute the program by pressing **init** followed by **go**. Examine the memory addresses and registers for the following results.

Memory Address	Register	Contents of Register	Memory Address	Contents of Address
1B00	a0	00	1A00	45
1B00	b0	00	1A01	88
1B01		1A	1A4F	2F
1B02	a1	50	1A50	79
1B02	b1	50		
1B03		1A		
1B04	a2	E5		

In this exercise, you have written data into memory by specifying both the data and the memory address in the instruction and by specifying the data and using the b register to point to addresses or to offsets of those addresses.

Also in this exercise, you have established the location of the a and b registers by setting the rp at the beginning of the program. The executive program does set the rp to an appropriate location for you, as in Exercise 1; however, it is good programming practice to set the rp yourself to avoid the random assignment of unsuitable memory space when you are not using the MAC Tutor. When you set the rp in a program, you should terminate the program with an illegal opcode instead of a return instruction. A return instruction causes the executive program to redefine the rp; therefore, when you examine the a and b registers, the data displayed will not correspond to your program.

Exercise 3 – Automatic Incrementing

This program exercise loads 8-bit data into memory using the b registers to point to addresses. The addresses are then automatically incremented.

Instruction	Hex. Code	Instruction Address	Comments
rp=1B00;	4D 0F 00 1B	1800 through 1803	Load rp with hex. address 1B00.
b0=1A00;	C0 0F 00 1A	1804 through 1807	Load register b0 (hex. addresses 1B00 and 1B01) with hex. address 1A00.
*b0++=5D;	83 0F 5D	1808 through 180A	Load 5D into hex. address specified by register b0 (1A00). Increment the contents of register b0 by 1. Register b0 now contains hex. address 1A01.

Exercise 3 (Continued)

Instruction	Hex. Code	Instruction Address	Comments
*b0++=96;	83 0F 96	180B through 180D	Load 96 into hex. address specified by register b0 (1A01). Increment the contents of register b0 by 1 (1A02).
b1=1A50;	C0 1F 50 1A	180E through 1811	Load register b1 (hex. addresses 1B02 and 1B03) with hex. address 1A50.
*b1++=AC;	83 1F AC	1812 through 1814	Load AC into hex. address specified by register b1 (1A50). Increment contents of register b1 by 1 (1A51).
*b1=22;	81 1F 22	1815 through 1817	Load hex. address specified by register b1 (1A51) with 22.
	FF	1818	Illegal opcode.

Execute the program and examine the memory addresses and registers for the following results.

Memory Address	Register	Contents of Register	Memory Address	Contents of Address
1B00	b0	02	1A00	5D
1B01		1A	1A01	96
			1A02	random
1B02	b1	51	1A50	AC
1B03		1A	1A51	22

In this exercise, you have loaded data into an address and then incremented that address in one instruction. This postincrementing of addresses provides a convenient way of storing data sequentially. Now that you know about postincrementing, try writing a shorter program for Exercise 1.

Exercise 4 — Loading 16-Bit Data Into Memory

This program exercise loads 16-bit data into memory by specifying the data and the address in the instruction and by using the registers to point to addresses, which are then automatically incremented.

Exercise 4 (Continued)

Instruction	Hex. Code	Instruction Address	Comments
rp=1B00;	4D 0F 00 1B	1800 through 1803	Load rp with hex. address 1B00.
b0=1A00;	C0 0F 00 1A	1804 through 1807	Load register b0 (hex. addresses 1B00 and 1B01) with hex.address 1A00.
*d0=8F40;	C1 0F 40 8F	1808 through 180B	Load 40 into hex. address specified by register b0 (1A00) and load 8F into the next hex. address (1A01). (Note that when the b registers are used to point to two successive addresses, they are referred to as d registers in the assembly language statements.)
*(d0+2)=91C6;	C2 0F 02 C6 91	180C through 1810	Load C6 into the hex. address two above the address specified by register b0 (1A02) and load 91 into the next hex. address (1A03).
b1=1A50;	C0 1F 50 1A	1811 through 1814	Load register b1 (hex. addresses 1B02 and 1B03) with hex. address 1A50.
*d1++=3456;	C3 1F 56 34	1815 through 1818	Load 56 into hex. address specified by register b1 (1A50) and load 34 into the next hex. address (1A51). Then, increment the contents of register b1 by 2 (1A52).
*d1=7629;	C1 1F 29 76	1819 through 181C	Load 29 into hex. address specified by register b1 (1A52) and load 76 into the next hex. address (1A53).
*1B04=479E;	C1 FF 04 1B 9E 47	181D through 1822	Load 9E into hex. address 1B04 and load 47 into hex. address 1B05. These addresses constitute register b2.
	FF	1823	Illegal opcode.

Exercise 4 (Continued)

Execute the program and examine the memory addresses and registers for the following results.

Memory Address	Register	Contents of Register	Memory Address	Contents of Address
1B00	b0	00	1A00	40
1B01		1A	1A01	8F
			1A02	C6
1B02	b1	52	1A03	91
1B03		1A	1A50	56
			1A51	34
1B04	b2	9E	1A52	29
1B05		47	1A53	76

In this exercise, you have used the b registers as pointers to two successive locations to store 16-bit data and, because the data comprise 16 bits, automatically incremented the addresses by 2. You have loaded the 16-bit data into memory by specifying the data and the address in the instruction. This instruction, located at hex. addresses 181D through 1822, requires six bytes and it is the longest of all the MAC-8 instructions.

Exercise 5 — Arithmetical and Logical Instructions

This program exercise performs arithmetical and logical operations on the corresponding bits of 8- or 16-bit data using the same addressing modes used for data transfer.

Instruction	Hex. Code	Instruction Address	Comments
rp=1B00;	4D 0F 00 1B	1800 through 1803	Load rp with hex. address 1B00.
b0=1A60;	C0 0F 60 1A	1804 through 1807	Load register b0 (hex. addresses 1B00 and 1B01) with hex. address 1A60.
*d0=59CE;	C1 0F CE 59	1808 through 180B	Load CE into hex. address specified by register b0 (1A60) and 59 into next hex. address (1A61).
b1=73CF;	C0 1F CF 73	180C through 180F	Load register b1 (hex. addresses 1B02 and 1B03) with 73CF. (This is 16-bit data, not an address.)

Exercise 5 (Continued)

Instruction	Hex. Code	Instruction Address	Comments
a2=a1;	80 21	1810, 1811	Move the contents of register a1 (CF) to register a2 (hex. address 1B04).
a2=a2Λa0;	88 20	1812, 1813	Perform the logical exclusive OR operation on the contents of register a2 (CF) and the contents of register a0 (60). Store the result (AF) in register a2.
b3=b0;	C0 30	1814, 1815	Move the contents of register b0 (1A60) to register b3 (hex. addresses 1B06 and 1B07).
b3=b3&b1;	D8 31	1816, 1817	Perform the logical AND operation on the contents of register b3 (1A60) and the contents of register b1 (73CF). (In register b3, hex. address 1A60 is used as 16-bit data.) Store the result (1240) in register b3.
b4=*d0;	C5 40	1818, 1819	Move the contents (CE and 59) of the hex. addresses specified by register b0 (1A60 and 1A61) to register b4 (hex. addresses 1B08 and 1B09).
b5=b1;	C0 51	181A, 181B	Move the contents of register b1 (73CF) to register b5 (hex. addresses 1B0A and 1B0B).
b5=b5+b4;	E8 54	181C, 181D	Perform the arithmetical add operation on the contents of register b4 (59CE) and the contents of register b5 (73CF). Store the result (CD9D) in register b5.
	FF	181E	Illegal opcode.

Exercise 5 (Continued)

Execute the program and examine the memory addresses and registers for the following results.

Memory Address	Register	Contents of Register	Memory Address	Contents of Address
1B00	a0	60	1A60	CE
1B00	b0	60	1A61	59
1B01		1A		
1B02	a1	CF		
1B02	b1	CF		
1B03		73		
1B04	a2	AF		
1B04	b2	AF		
1B05		random		
1B06	a3	40		
1B06	b3	40		
1B07		12		
1B08	a4	CE		
1B08	b4	CE		
1B09		59		
1B0A	a5	9D		
1B0A	b5	9D		
1B0B		CD		

In this exercise you have performed the logical exclusive OR and AND operations and the arithmetical add operation. In the exclusive OR operation, located at hex. addresses 1812 and 1813, the corresponding data bits are examined. If either of the bits is a 1, the resulting bit is a 1. If both of the bits are 1s or 0s, the resulting bit is a 0 as shown in the following.

Hex. Code	Binary Equivalent
CF	1100 1111
60	0110 0000
	Logical exclusive OR result =
AF	1010 1111

In the logical AND operation, located at hex. addresses 1816 and 1817, the corresponding data bits are examined, and if both bits are 1s, the resulting bit is a 1. If both bits are 0s or if one bit is a 1 and the other bit is a 0, the resulting bit is a 0, as shown in the following.

Exercise 5 (Continued)

Hex. Code	Binary Equivalent
1A60	0001 1010 0110 0000
73C7	0111 0011 1100 0111
	Logical AND result =
1240	0001 0010 0100 0000

In the arithmetical add operation, located at hex. addresses 181C and 181D, the bits are added according to the following.

Bit Values	Carry In	Sum	Carry Out
0 + 0	0	0	0
0 + 1	0	1	0
1 + 0	0	1	0
1 + 1	0	0	1
0 + 0	1	1	0
0 + 1	1	0	1
1 + 0	1	0	1
1 + 1	1	1	1

The following example shows the resulting bits.

Hex. Code	Binary Equivalent
73CF	0111 0011 1100 1111
59CE	0101 1001 1100 1110
	Arithmetical add result =
CD9D	1100 1101 1001 1101

Exercise 6 — Addition of 8-Bit Number to 16-Bit Number

This program exercise illustrates the distinction between signed and unsigned numbers.

Instruction	Hex. Code	Instruction Address	Comments
rp=1800;	4D 0F 00 1B	1800 through 1803	Load rp with hex. address 1B00.
b0=28A0;	C0 0F A0 28	1804 through 1807	Load register b0 (hex. addresses 1B00 and 1B01) with 28A0 (16-bit data).
a1=77;	80 1F 77	1808 through 180A	Load register a1 (hex. address 1B02) with 77 (8-bit data).
b0=b0+a1;	7D 01	180B, 180C	Perform the arithmetical add operation on the contents of register b0 (28A0) and the contents of register a1 (77). Store the result (2917) in register b0.
	FF	180D	Illegal opcode.

Execute the program and examine the memory addresses and registers for the following results.

Memory Address	Register	Contents of Register
1B00	a0	17
1B00	b0	17
1B01		29
1B02	a1	77

Repeat the program with the following change.

Instruction	Hex. Code	Instruction Address	Comments
b0=b0+logical a1;	75 01	180B, 180C	Perform the logical add operation instead of the arithmetical add operation on the contents of register b0 (28A0) and the contents of register a1 (77). Store the result (2917) in register b0.

This change is made by changing the contents of hex. address 180B from 7D to 75.

After execution, the results are as follows.

Memory Address	Register	Contents of Register
1B00	a0	17
1B00	b0	17
1B01		29
1B02	a1	77

Repeat the program with the following changes.

Instruction	Hex. Code	Instruction Address	Comments
a1=C7;	80 1F C7	1808 through 180A	Load register a1 (hex. address 1B02) with C7 (equal to −39 in 2s complement notation).
b0=b0+a1;	7D 01	180B, 180C	Perform arithmetical add operation on the contents of register b0 (28A0) and the contents of register a1 (C7). Store the result (2867) in register b0.

These changes are made by changing the contents of hex. address 180A from 77 to C7 and the contents of hex. address 180B from 75 to 7D.

After execution, the results are as follows.

Memory Address	Register	Contents of Register
1B00	a0	67
1B00	b0	67
1B01		28
1B02	a1	C7

Repeat the program with the following change.

Exercise 6 (Continued)

Instruction	Hex. Code	Instruction Address	Comments
b0=b0+logical a1;	75 01	180B, 180C	Perform the logical add operation on the contents of register b0 (28A0) and the contents of register a1 (C7). Store the result (2967) in register a1.

This change is made by changing the contents of hex. address 180B from 7D to 75. After execution, the results are as follows.

Memory Address	Register	Contents of Register
1B00	a0	67
1B00	b0	67
1B01		29
1B02	a1	C7

In this exercise, you have performed the arithmetical and the logical add operations of an 8-bit number to a 16-bit number. An 8-bit number can be an unsigned number or a signed 2s complement number (one sign bit and 7-bit magnitude). The arithmetical add operation treats 8-bit numbers as signed 2s complement numbers. For 2s complement numbers, the format is as follows.

Most Significant Bit (Sign Bit)	Sign	Value
0	+	True magnitude.
1	–	2s complement of true magnitude.

The first part of this exercise contains the arithmetical add operation of the 8-bit number 0111 0111 (hex. 77). The sign bit is 0; therefore, the signed 2s complement is +77, as shown in the following.

Hex. Code	Binary Equivalent
28A0	0010 1000 1010 0000
+77	111 0111
	Arithmetical add result =
2917	0010 1001 0001 0111

Exercise 6 (Continued)

The second part of this exercise contains the logical add instruction of the 8-bit number 0111 0111 (hex. 77). This instruction treats 8-bit numbers as unsigned numbers; therefore, 0111 0111 is a positive quantity, and the results are the same as the results in the first part of the exercise.

The third part of this exercise contains the arithmetical add instruction of the 8-bit number 1100 0111 (hex. C7). The sign bit is 1; therefore, the signed 2s complement is –39. Generally, this is obtained in the following manner.

$$N_{\text{2s complement}} = \sim N + 1$$

or

$$N = \sim N_{\text{2s complement}} + 1$$

where $\sim$ indicates the bit-by-bit complement of the number and N is the true magnitude. In this exercise,

Sign bit = 1

$$N_{\text{2s complement}} = 100\ 0111$$

$$\sim N_{\text{2s complement}} = 011\ 1000$$

N = 011 1001 = 39 (hex.)

Hex. Code	Binary Equivalent
28A0	0010 1000 1010 0000
–39	–011 1001

Arithmetical add result =

2867	0010 1000 0110 0111

To obtain the above result, the equivalent of subtraction is performed. The bits are subtracted according to the following.

Exercise 6 (Continued)

Bit Values	Borrow In	Difference	Borrow Out
0 – 0	0	0	0
0 – 1	0	1	1
1 – 0	0	1	0
1 – 1	0	0	0
0 – 0	1	1	1
0 – 1	1	0	1
1 – 0	1	0	0
1 – 1	1	1	1

The last part of the exercise contains the logical add instruction of the 8-bit number 1100 0111 (hex. C7). Again, the logical add instruction treats 8-bit numbers as unsigned; therefore, 1100 0111 is a positive quantity. This is shown in the following example.

Hex. Code	Binary Equivalent
28AC	0010 1000 1010 0000
C7	1100 0111
	Logical add result =
2967	0010 1001 0110 0111

Exercise 7 – Condition Register Observation

This program exercise permits observation of the condition register (cr) by moving, or pushing, its contents onto the memory stack.

Instruction	Hex. Code	Instruction Address	Comments
rp=1B00;	4D 0F 00 1B	1800 through 1803	Load rp with hex. address 1B00.
sp=1BA0 ;	0D 0F A0 1B	1804 through 1807	Load stack pointer (sp) with hex. address 1BA0.
a0=0;	20 00	1808, 1809	Load register a0 with all 0s. Now, the cr contains 02.

Exercise 7 (Continued)

Instruction	Hex. Code	Instruction Address	Comments
push(a0);	06 00	180A, 180B	Move the contents of register a0 (00) to the hex. address one below the hex. address specified by the sp (1B9F) and decrement the sp by 1.
push(cr);	07	180C	Move the contents of the cr (02) to the hex. address one below the hex. address specified by the sp (1B9E) and decrement the sp by 1.
--a0;	28 08	180D, 180E	Decrement the contents of register a0 by 1. Now, register a0 contains FF (−1 in 2s complement notation) and the cr contains 39.
push(a0);	06 00	180F, 1810	Move the contents of register a0 (FF) to the hex. address one below the hex. address specified by the sp (1B9D) and decrement the sp by 1.
push(cr);	07	1811	Move the contents of the cr (39) to the hex. address one below the hex. address specified by the sp (1B9C) and decrement the sp by 1.
a0=−a0;	24 00	1812, 1813	Negate the contents of register a0. Register a0 now contains 01 (+1) and the cr contains 28.
push(a0);	06 00	1814, 1815	Move the contents of register a0 (01) to the hex. address one below the hex. address specified by the sp (1B9B) and decrement the contents of the sp by 1.

Exercise 7 (Continued)

Instruction	Hex. Code	Instruction Address	Comments
push(cr);	07	1816	Move the contents of the cr (28) to the hex. address one below the hex. address specified by the sp (1B9A) and decrement the sp by 1.
a0=a0>>>1;	34 0F	1817, 1818	Rotate the contents of register a0 one bit to the right. This moves the least significant bit (LSB) to the most significant bit (MSB) and also to the carry bit of the cr. Register a0 now contains 80 and the cr contains 09.
push(a0);	06 00	1819, 181A	Move the contents of register a0 (80) to the hex. address one below the hex. address specified by the sp (1B99) and decrement the sp by 1.
push(cr);	07	181B	Move the contents of the cr (09) to the hex. address one below the hex. address specified by the sp (1B98) and decrement the contents of the sp by 1.
	FF	181C	Illegal opcode.

Execute the program and examine the memory addresses for the following results.

Memory Address	Contents	Condition Register Bits							
		Flag	Enable	Odd	Ones	Carry	Ovfl	Zero	Neg
1B98 (final sp)	09	0	0	0	0	1	0	0	1
1B99	80								
1B9A	28	0	0	1	0	1	0	0	0
1B9B	01								
1B9C	39	0	0	1	1	1	0	0	1
1B9D	FF								
1B9E	02	0	0	0	0	0	0	1	0
1B9F	00								
1BA0 (initial sp)	random								

Exercise 7 (Continued)

As described previously, the stack is often used to store return addresses when subroutine call instructions are executed. However, in this exercise, you are using the stack to store the results of various operations that affect the cr. This permits you to observe the status of the individual bits of the cr. The bits are set to 1 according to the following.

- The zero bit is set to 1 when the operation results in all 0s.
- The ones bit is set to 1 when the operation results in all 1s.
- The odd bit is set to 1 when the operation results in the LSB equal to 1.
- The neg bit is set to 1 when the operation results in the MSB equal to 1 (MSB represents sign bit in 2s complement notation).
- The carry bit is affected by the decrement, negate, and rotate instructions. The decrement operation results in a negative number and the carry bit is set to 1, indicating a borrow in this case. The carry bit remains set to 1 after the negate operation, even though the result is positive (+1), because the negate operation involves subtracting the specified number (which is in 2s complement form) from 0, yielding an apparent borrow. The carry bit remains set to 1 by the rotate 8 instruction because the LSB is rotated into the carry bit as well as into the MSB. As illustrated by this exercise, the carry bit result is not always obvious and should be used carefully.

When register a0 is loaded with 00 (0000 0000), the zero bit is set to 1 and the cr contains 02 (0000 0010). When the contents of register a0 are decremented to −1 (1111 1111), the odd, ones, carry, and neg bits are set to 1s and the cr contains 39 (0011 1001). When the contents of register a0 are negated and become +1 (0000 0001), the odd and carry bits are set to 1s and the cr contains 28 (0010 1000). And, when the contents of register a0 are rotated and become 80 (1000 0000), the carry and the neg bits are set to 1s and the cr contains 09 (0000 1001).

Exercise 8 — Multiplication of Two 8-Bit Unsigned Numbers

This program exercise breaks down the multiplication operation into a series of addition and shifting operations that can be executed by the MAC-8 microprocessor.

Instruction	Hex. Code	Instruction Address	Comments
rp=1B00;	4D 0F 00 1B	1800 through 1803	Load rp with hex. address 1B00.
a0=8;	80 0F 08	1804 through 1806	Load register a0 (hex. address 1B00) with 8. Register a0 is being used as a counter to store the number of bits in the multiplier.

Exercise 8 (Continued)

Instruction	Hex. Code	Instruction Address	Comments
b1=0003;	C0 1F 03 00	1807 through 180A	Load register b1 (hex. addresses 1B02 and 1B03) with the multiplicand 03. Since the number is limited to 8 bits, the high-order byte contains all 0s.
a2=02;	80 2F 02	180B through 180D	Load register a2 (hex. address 1B04) with the multiplier 02.
b3=0;	60 30	180E, 180F	Load register b3 (hex. addresses 1B06 and 1B07) with all 0s. The result of the multiplication, a 16-bit number, is stored in this register.
check: if(!bit(0,a2)) goto next;	52 20 03	1810 through 1812	If bit 0 (LSB) of the multiplier (02) is 0, go to the instruction next (three bytes ahead).
b3=b3+b1;	E8 31	1813, 1814	Add the contents of register b1 to the contents of register b3 and store the result in register b3. The multiplicand is being added to the partial product only when bit 0 of the multiplier is 1.
next: a2=a2>>>1;	34 2F	1815, 1816	Rotate the contents of register a2 one bit to the right. This checks each multiplier bit in sequence.
b1=b1+b1;	E8 11	1817, 1818	The contents of register b1 are added to themselves and the result is stored in register b1. This has the effect of shifting the multiplicand one bit to the left.
--a0;	28 08	1819, 181A	Decrement the contents of register a0 by 1 and store the result in register a0.

Exercise 8 (Continued)

Instruction	Hex. Code	Instruction Address	Comments
if (!zero) goto check;	41 F1 10 18	181B through 181E	If the previous instruction results in a nonzero condition, go to the instruction check (hex. address 1810).
	FF	181F	Illegal opcode.

Execute the program and examine the memory addresses and registers for the following results.

Memory Address	Register	Contents of Register	Function
1B00	a0	00	Counter.
1B02	b1	00	Original multiplicand location.
1B03		03	Final multiplicand location.
1B04	a2	02	Multiplier.
1B06	b3	06	LO product.
1B07		00	HI product.

In this exercise, you have used conditional transfer instructions to repeatedly execute the adding and shifting that make up the multiplication operation. Decrementing register a0 affects the zero bit and, because initially register a0 contains the number 8 (the number of bits to be multiplied), the instructions starting at check are executed eight times.

The MAC-8 assembly language permits the use of labels to identify specific memory addresses to which a program can branch. Labels eliminate the need to assign absolute addresses when an assembler program is used to generate the machine code. Labels also make assembly language programs more readable. In this program, the label check is equivalent to *1810 and the label next is equivalent to *1815.

This program can be repeated, using different numbers, simply by changing the program byte at hex. address 1809, which defines the multiplicand, and the program byte at hex. address 180D, which defines the multiplier. The product is found in register b3 (at hex. addresses 1B06 and 1B07).

Exercise 8 (Continued)

Some additional multiplication exercises that you can verify are as follows.

Multiplicand	Multiplier	Product
67	85	3583
D6	AD	909E
FF	FF	FE01
00	xx	0000
xx	00	0000

Note: xx are random contents.

The multiplicand and the multiplier can be specified by the program, as in the above exercises, or they can be preloaded in specific memory locations and addressed by the program. This is another exercise that you can perform.

Exercise 9 — Memory Block Transfer

This program exercise moves a block of data from one area of memory to another. To execute this program, you have to preload the old starting address, the new starting address, and the size of the data block.

Instruction	Hex. Code	Instruction Address	Comments
rp=1B00;	4D 0F 00 1B	1800 through 1803	Load rp with hex. address 1B00.
move: a3=*b0++;	87 30	1804, 1805	Move the contents of the hex. address specified by register b0 to register a3. Increment the contents of register b0 by 1.
*b1++=a3;	83 13	1806, 1807	Move the contents of register a3 to the hex. address specified by register b1. Increment the contents of register b1 by 1.
--b2;	68 28	1808, 1809	Decrement the contents of register b2 by 1 and store the result in register b2.

Exercise 9 (Continued)

Instruction	Hex. Code	Instruction Address	Comments
if(!zero) goto move;	41 F1 04 18	180A through 180D	If the previous instruction resulted in a nonzero condition, go to the instruction move (hex. address 1804).
	FF	180E	Illegal opcode.

Now, preload the necessary data as follows.

Hex. Code	Address	Comments
A0 19	1B00,1B01	Load the old starting hex. address, 19A0, into hex. addresses 1B00 and 1B01, which constitute register b0.
B0 19	1B02,1B03	Load the new starting hex. address, 19B0, into hex. addresses 1B02 and 1B03, which constitute register b1.
0C 00	1B04,1B05	Load the number of bytes to be be moved into hex. addresses 1B04 and 1B05, which constitute register b2.

Alternatively, you can preload the b registers by keying in the following sequence.

/b = 19A0 + 19B0 + 000C

The first instruction of the program then assigns specific memory addresses to these registers. A disadvantage of preloading (either way) is that the numbers change when the program is executed. If you want to rerun the program, even using the same starting addresses, you must preload the numbers again.

Execute the program and observe that the block of data that is stored at hex. addresses 19A0 through 19AB is stored also at hex. addresses 19B0 through 19BB.

In this exercise, you have moved a block of data from one area of memory to another, but note that the block of data still exists in the old area of memory. You can move any size block; the only restrictions are that you have to use the existent RAM and that the area to which you are transferring a block of data cannot overlap the area from which the block of data is being transferred.

Exercise 9 (Continued)

Another way to move a block of data is to call the subroutine move ( ). This subroutine is stored permanently as a part of the executive program, so you do not have to enter it through the keypad. You can verify its existence by examining the contents of hex. addresses 022F through 023A. Also, any attempt to change the contents of these addresses will fail because you are addressing the ROM rather than the RAM.

The subroutine move ( ) is as follows.

	Instruction	Hex. Code	Instruction Address	Comments
move:	set(02);	01 02	022F, 0230	Set zero bit to overcome hardware error in early version MAC-8 microprocessors.
	b9–b10;	F0 9A	0231, 0232	
	if(zero)return;	64 01	0233, 0234	Compare addresses and return to another point in the executive program when data transfer is complete.
	a0=*b9++;	87 09	0235, 0236	This instruction and the next transfer the data byte.
	*b8++=a0;	83 80	0237, 0238	
	goto move;	58 F4	0239, 023A	Go to instruction move.

The new starting address is stored in register b8. The starting address of the first byte to be moved is stored in register b9. Finally, the address one above that of the last byte to be moved is stored in register b10. To execute the same memory block transfer described in the first part of this exercise, key in only the following.

Key Sequence	Comments
/b8=19B0	Load register b8 with new starting address.
+19A0	Load register b9 with old starting address.

Exercise 9 (Continued)

Key Sequence	Comments
+19AC	Load register b10 with the address one above that of the last data byte to be moved.
*022F	Go to first address of move ( ).
go	Execute subroutine.

INPUT/OUTPUT PROGRAM EXERCISES

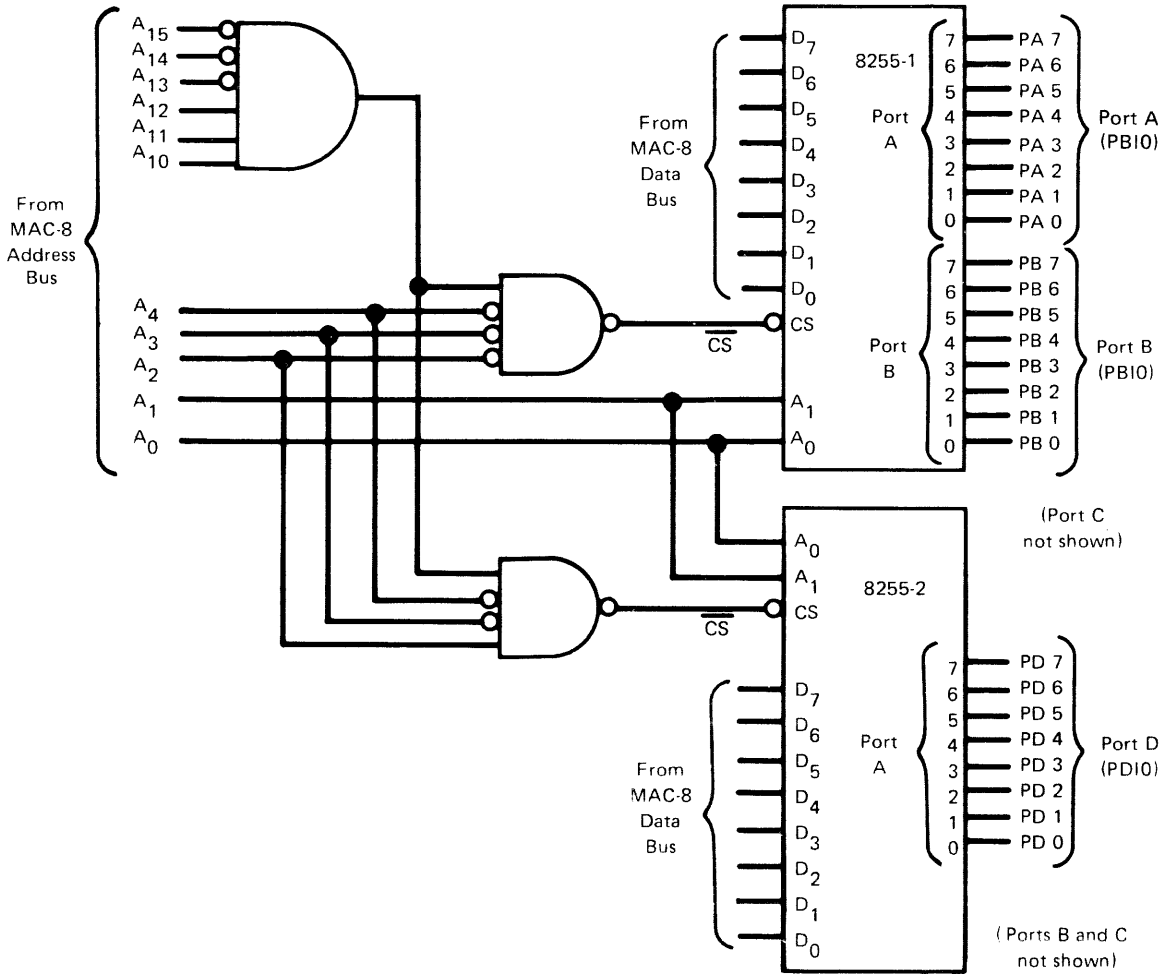
The following exercises demonstrate the ability to program the MAC Tutor to communicate with input/output (I/O) devices. When using the MAC Tutor alone, the input device is the keypad and the output device is the display. The executive program, upon input instructions from the keypad, reads the keypad and presents output signals to the LED segments. This is possible because hardware connections associate the keypad and LED arrays with specific memory locations that can be addressed by software. As a result, each key is assigned a specific function, allowing you to display and change memory and register contents, as well as execute programs.

In these exercises, you will learn how to access the keypad, using software to define key functions, and display arbitrary patterns on the 7-segment LED arrays.

BRIEF HARDWARE DESCRIPTION

I/O Address Decoding Circuitry

The main hardware elements are two Intel 8255 programmable peripheral interface chips (8255-1 and 8255-2) with I/O port pins connected to the keypad and display. The chips are activated when valid address bits are presented to the MAC-8 microprocessor address bus. (See the I/O Address Decoding Circuitry Schematic Diagram, which shows a portion of the complex decoding circuitry.)



I/O Address Decoding Circuitry Schematic Diagram

The 8255 chip select input ($\overline{CS}$) must be a logical 0 to initiate operation. The valid address bits are as follows.

	Valid Address Bits															
Chip	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
8255-1	0	0	0	1	1	1	X	X	X	X	X	0	0	0	X	X
8255-2	0	0	0	1	1	1	X	X	X	X	X	0	0	1	X	X

Note: X can be logical 1 or 0.

If address bits A_9 and A_8 are logical 1s and A_7, A_6, and A_5 are logical 0s, the I/O address space is limited to hex. addresses 1F00 through 1F03 for the 8255-1 chip and 1F04 through 1F07 for the 8255-2 chip. In addition, the value of address bits A_1 and A_0 determine the selection of one of three 8255 chip I/O ports (A, B, or C) or an internal control word register.

The address bit values for port and register selection are as follows.

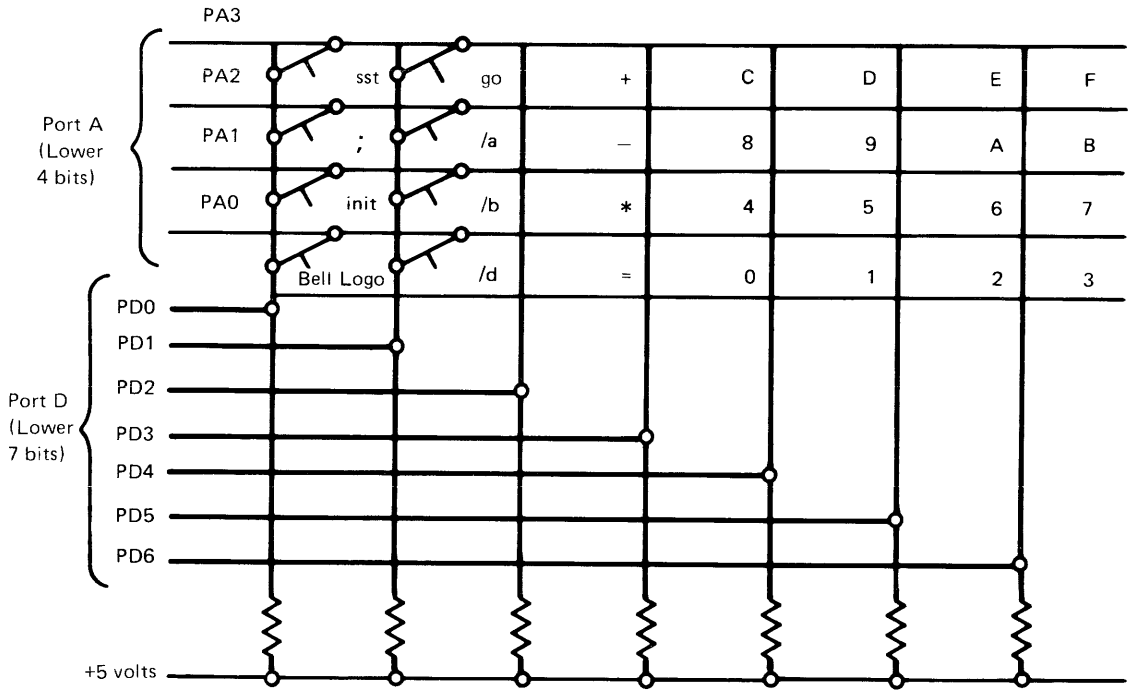
A_1	A_0	Port/Register Selection
0	0	Port A
0	1	Port B
1	0	Port C
1	1	Control word register.

Note: As indicated, the eight MAC-8 data bus lines will be connected to port A, B, or C or to an internal 8255 chip control word register.

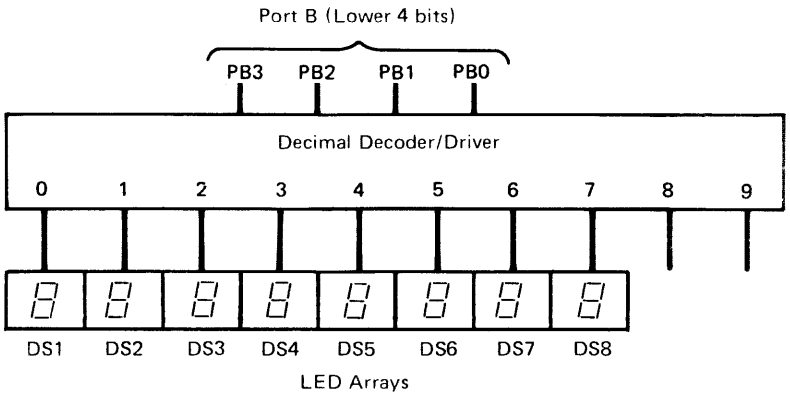
The control word register for the 8255-1 chip is designated PCNTRL and the control word register for the 8255-2 chip is designated QCNTRL. Also, as indicated on the I/O Address Decoding Schematic Diagram, the port A pins on the 8255-2 chip are externally designated as port D pins (PD7 through PD0). This is necessary to distinguish them from the port A pins on the 8255-1 chip. These designations are also used in other MAC Tutor hardware and software documentation.

Keypad/Display Circuitry

Ports A, B, and D are connected to the keypad and display as shown in the following diagrams.



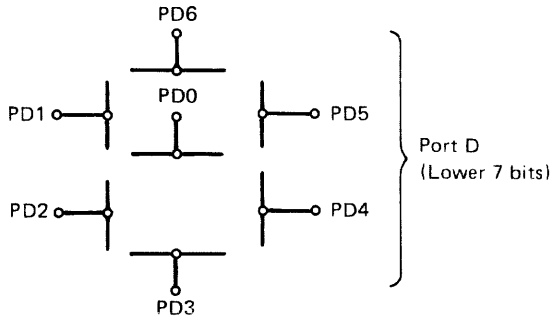
Keypad Circuitry Schematic Diagram



LED Array Select Circuitry

LED Array Inputs and Outputs

Decoder Input				Digit Selected
PB3	PB2	PB1	PB0	
0	0	0	0	DS1
0	0	0	1	DS2
0	0	1	0	DS3
0	0	1	1	DS4
0	1	0	0	DS5
0	1	0	1	DS6
0	1	1	0	DS7
0	1	1	1	DS8



Notes:

1. Logical 1 lights LED.
2. Same circuitry for all eight LED segments.

Segment Circuitry Diagram

Memory Locations

The I/O address decoding and keypad/display circuitry establish the following memory allocations.

Memory Address	Contents	Function
1F00	PAIO	Keypad row select.
1F01	PBIO	LED array select (lower four bits).
1F03	PCNTRL	Control word register for ports A and B.
1F04	PDIO	Keypad column select and display segment select (lower seven bits).
1F07	QCNTRL	Control word register for port D.

Ports A, B, and D are either input or output ports depending on the number stored in the control word register of the appropriate chip. A number is stored in the control word register by loading it, via software, into the associated memory location (hex. address 1F03 for the 8255-1 chip or hex. address 1F07 for the 8255-2 chip). Upon execution of the proper instruction, the number is transferred from the MAC-8 data bus into the appropriate control register. The number is held in this register until another software instruction changes it. The control word register requirements for ports A and B of the 8255-1 chip and port D of the 8255-2 chip are as follows.

Control Word Register – PCNTRL								Port Direction	
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Port A	Port B
1	0	0	1	X	0	1	X	Input	Input
1	0	0	1	X	0	0	X	Input	Output
1	0	0	0	X	0	1	X	Output	Input
1	0	0	0	X	0	0	X	Output	Output

Control Word Register – QCNTRL								Port Direction
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Port D
1	0	0	1	X	0	X	X	Input
1	0	0	0	X	0	X	X	Output

Notes:

1. X can be logical 1 or 0.
2. The CS input must be logical 0 and the address bits A_1 and A_0 must be logical 1s.

MAC-8 microprocessor data are received from an input port and are sent to an output port. The display circuitry requires that data be sent to port B to select the appropriate LED array and sent to port D to light the desired segment of each LED array. (The status of port A is immaterial at this time.) The control word register content for the 8255-1 chip (PCNTRL) can be set to either 1001 X00X or 1000 X00X to define port B as an output port. The control word register content for the 8255-2 chip (QCNTRL) can be set to either 1000 X01X or 1000 X00X to define port D as an output port. As a result, PCNTRL is set to 1000 0000 (80 hex.) and QCNTRL is set to 1000 0000 (80 hex.) for the display routines.

The keypad control requirements are dependent on the type of program. A simple way of providing keypad control is to write a keypad scanning routine that examines each row of keys in sequence. When a signal is provided to a particular row, a depressed key in that row will transfer the signal to the line for the column corresponding to that key. In this manner, a depressed key can be sensed by identifying the row and column location. The Keypad Circuitry Schematic Diagram shows that this is accomplished by defining port A as an output and port D as an input. The port D lines are at a logical 1 level when no keys are depressed. If a logical 0 level is presented (via an instruction) to a row select line, a key closure in that row results in a logical 0 level only on the column select line corresponding to that key. If the column select number on port D is sent to the MAC-8 microprocessor, a logical 0 level on the row and column line defines a unique key and the appropriate action is taken. (It should be noted from the Keypad Circuitry Schematic Diagram that the keypad row and column numbers differ from port A and D bit numbers.) It is also desirable to define port B as an input during keypad scanning to prevent activating the display. For the keypad read routine, PCNTRL is set to 1000 X01X and QCNTRL is set to either 1001 X01X or 1001 X00X. As a result, PCNTRL is set to 1000 0010 (82 hex.) and QCNTRL is set to 1001 0000 (90 hex.).

Some shortcuts have been taken to reduce the number of instruction bytes that have to be entered for the I/O programs. For example, instead of initializing the rp (as in previous program exercises), the value is set by the executive program. Also, certain b registers must be preloaded. In the previous exercises, registers were loaded under program control (instructions were included within the program). This necessitated added instructions, but resulted in a self-contained program. In the I/O program exercises, the preloaded b registers contain addresses that are not changed by the program. Therefore, the program can be rerun without reloading those registers. After executing each I/O program, it is necessary to press the RESET button to allow the executive program to regain control (an exception is where control is automatically returned to the executive program as in Exercise 12). In addition, now that you are reasonably familiar with the MAC-8 instructions, the comments accompanying the I/O programs are less detailed.

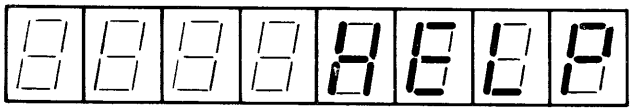
Exercise 10 – Fixed Message Display

This program exercise demonstrates the ability to control the lighting of individual segments of the LED arrays and to use software to continually refresh the display so that it remains visible.

Exercise 10 (Continued)

Instruction	Hex. Code	Instruction Address	Comments
fixed: *(b0+2)=80;	82 0F 02 80	1800 through 1803	Set value of PCNTRL to define port B as output port for LED array select.
*(b0+6)=80;	82 0F 06 80	1804 through 1807	Set value of QCNTRL to define port D as output port for segment select.
start: b1=1900;	C0 1F 00 19	1808 through 180B	Store address of leftmost LED array display code.
*b0=0;	21 00	180C, 180D	Set PBIO to select leftmost LED array.
light: *(b0+3)=*(b1+0);	C4 01 00 03	180E through 1811	SET PDIO to output display code.
(b0+3)=(b1+0);	C4 01 00 03	1812 through 1815	Repeat to brighten display.
*(b0+3)=0;	22 00 03	1816 through 1818	Blank display before selecting next LED array.
++b1;	68 10	1819, 181A	Display next LED array to the right.
++*b0;	29 00	181B, 181C	
*b0−8;	B1 0F 08	181D through 181F	After all eight arrays are displayed, repeat the display for continuous refreshing.
if(lt) goto light;	49 F8 0E 18	1820 through 1823	
goto start;	59 08 18	1824 through 1826	

Now, key in /b = 1 F 0 1 to preload the port B address and * 1 9 0 0 = 0 0 + 0 0 + 0 0 + 0 0 + 3 7 + 4 F + 0 E + 6 7 to store codes in hex. addresses 1900 through 1907. Press init and go. You should observe the following.



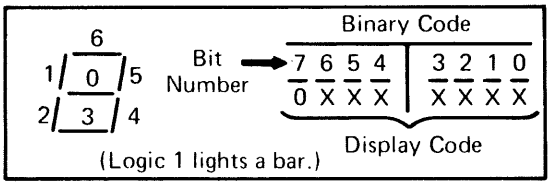
The four leftmost LED arrays are blank because you entered 0s at hex. addresses 1900 through 1903. Leave this program in memory for use in the next exercise. Press RESET to allow the executive program to regain control.

Exercise 10 (Continued)

With this program you have used the LED arrays as output devices that can be addressed as memory locations to display a particular message. The display must be blanked after each segment is selected because only one set of signals required for the display can be stored. If the display is not blanked, each segment will carry over to the next. The brightness of the message is dependent on the proportion of time that the message is on compared to the time that the display is blank. This program allows control of the individual segments of the LED arrays so you are not limited to displaying hex. digits. In fact, you are not limited to displaying the message HELP. All you have to do to change the message is to change the contents of hex. addresses 1900 through 1907. Try changing the message using the following list of codes for displaying alphanumerics.

Digit Display	Hex. Code
0	7E
1	30
2	6D
3	79
4	33
5	5B
6	5F
7	70
8	7F
9	73

Letter Display	Hex. Code
A	77
B	1F
C	0D
D	3D
E	4F
F	47
G	5E
H	37
I	10
J	3C
L	0E
N	15
O	1D
P	67
R	05
U	1C
Y	33



Note: These codes can be entered into hex. addresses 1900 through 1907 for Exercise 10 and into addresses 1901, 1903, 1905, 1907, etc., for Exercise 13.

Exercise 11 – Keypad Scan Routine

This program exercise makes use of the keypad as an input device and action is initiated when a specific key is depressed. (This program is loaded at hex. address 1A00 and is dependent on the fixed message display program exercise.)

Exercise 11 (Continued)

Instruction	Hex. Code	Instruction Address	Comments
*(b0+2)=82;	82 0F 02 82	1A00 through 1A03	Define port A as output port (for keypad row select).
*(b0+6)=90;	82 0F 06 90	1A04 through 1A07	Define port D as input port (for keypad column select).
a2=FE;	80 2F FE	1A08 through 1A0A	Prepare to check first row for depressed key.
loop: *(b0−1)=a2;	82 02 FF	1A0B through 1A0D	Output signal to row.
if(bit(0,a2))goto next;	5A 20 0A	1A0E through 1A10	Row 4 key down?
a3=*(b0+3);	86 30 03	1A11 through 1A13	If so, input signal from column.
if(bit(1,a3)) goto next;	5A 31 04	1A14 through 1A16	Is column 2 key down?
fixed( );	79 00 18	1A17 through 1A19	Display fixed message of previous exercise only if key /d is depressed.
next: a2=a2 <<<1;	34 21	1A1A, 1A1B	Check next row and repeat.
goto loop;	59 0B 1A	1A1C through 1A1E	

Note: Fixed( ) is the label for Exercise 10.

Key in *** 1 A 0 0** and press **go**. Observe that the display is blank. Press **/d**. You should observe the following.

In this exercise, you have used the functioning of a key to close a circuit between the row and column corresponding to that key. The program issues a signal to each row in sequence and is able to detect a specific key closure by determining if the signal has been transmitted

to the column corresponding to the key. Instructions are executed in microseconds so the keypad scan is rapid enough to catch any key closure. The key functions are under control of this program rather than the executive program; therefore, the key labels are meaningless and any key can be used to initiate the message display. The row selection information is part of the instruction byte stored at hex. address 1A0F and the column selection information is stored at hex. address 1A15.

Exercise 12 – Sequential Display of Segment Patterns

This program exercise causes a sequential display of 128 different patterns using a single 7-segment LED array. A delay loop has been included in the program to control the duration of the pattern.

Instruction	Hex. Code	Instruction Address	Comments
*(b0+2)=80;	82 0F 02 80	1800 through 1803	Define port B as output port (for LED array select).
*(b0+6)=80;	82 0F 06 80	1804 through 1807	Define port D as output port (for segment select).
a2=0;	20 20	1808, 1809	Initialize display pattern.
*b0=07;	81 0F 07	180A through 180C	Select rightmost array for display.
light: b4=*d3;	C5 43	180D, 180E	Store delay time bytes.
*(b0+3)=a2;	82 02 03	180F through 1811	Display pattern.
loop: --b4;	68 48	1812, 1813	Hold display for specified time.
if(!zero) goto loop;	41 F1 12 18	1814 through 1817	
++a2;	28 20	1818, 1819	Prepare for next pattern.
if(bit(7,a2)) goto finish;	5A 27 04	181A through 181C	After all 128 patterns are displayed, return to executive program. This eliminates the need to press RESET.
goto light;	59 0D 18	181D through 181F	
finish: return;	66	1820	

Exercise 12 (Continued)

Key in * 1 8 F D = 0 0 + 5 0 to load a number relating to a delay time of approximately 0.5 second per pattern; /b = 1 F 0 1 to preload the port B address; and /b 3 = 1 8 F D to preload the delay time address. Press **init** and **go** and observe the sequential display of 128 patterns.

In this exercise, you have repeatedly incremented the segment select byte to display all the possible patterns. You have also used a register as a counter to provide a controlled timing loop by branching when a preassigned count is reached. You can speed up or slow down the pattern sequence by decreasing or increasing, respectively, the number that you have loaded in hex. addresses 18FD and 18FE.

Exercise 13 – Cyclic Display

This program exercise extends the scope of the program contained in the previous exercise by displaying an arbitrary sequence of patterns using any or all of the 7-segment LED arrays. The programmed delay loop is used to alternately light and blank the display.

Instruction	Hex. Code	Instruction Address	Comments
*(b0+2)=80;	82 0F 02 80	1800 through 1803	Define port B as output port (for LED array select).
*(b0+6)=80;	82 0F 06 80	1804 through 1807	Define port D as output port (for segment select).
start: b1=1900;	C0 1F 00 19	1808 through 180B	Store starting address of pattern.
a3=*b2;	85 32	180C, 180D	Save number of patterns per cycle.
next: a4=*b1++;	87 41	180E, 180F	Select LED array and prepare to output segment select code stored in next address.
*b0=a4;	81 04	1810, 1811	
dsply( );	79 20 18	1812 through 1814	Go to display subroutine.
++b1;	68 10	1815, 1816	Prepare for next display in pattern.
--a3;	28 38	1817, 1818	
if(!zero) goto next;	41 F1 0E 18	1819 through 181C	Repeat pattern.

Exercise 13 (Continued)

Instruction	Hex. Code	Instruction Address	Comments
goto start;	59 08 18	181D through 181F	
dsply: *(b0+3)=*(b1+0);	C4 01 00 03	1820 through 1823	Display and hold pattern using delay subroutine.
delay( );	79 2E 18	1824 through 1826	
*(b0+3)=0;	22 00 03	1827 through 1829	Blank pattern and hold using delay subroutine.
delay( );	79 2E 18	182A through 182C	
return;	66	182D	Return to main program.
delay: b6=*d5;	C5 65	182E, 182F	Hold pattern or blank display for specified time.
loop: --b6;	68 68	1830, 1831	
if(!zero) goto loop;	41 F1 30 18	1832 through 1835	
return;	66	1836	Return to display routine.

Key in * 1 8 F D = 0 0 + 0 C + 1 4 to load a delay time of approximately 5 seconds per cycle and the number of patterns per cycle. Key in the following LED array and segment selection information.

Hex. Addresses	LED Array (Low Address)	Segment (High Address)
1900, 1901	00	40
1902, 1903	01	40
1904, 1905	02	40
1906, 1907	03	40
1908, 1909	04	40
190A, 190B	05	40
190C, 190D	06	40
190E, 190F	07	40
1910, 1911	07	20
1912, 1913	07	10
1914, 1915	07	08
1916, 1917	06	08
1918, 1919	05	08
191A, 191B	04	08
191C, 191D	03	08

Hex. Addresses	LED Array (Low Address)	Segment (High Address)
191E, 191F	02	08
1920, 1921	01	08
1922, 1923	00	08
1924, 1925	00	04
1926, 1927	00	02

Now key in /b = **1 F 0 1** to preload the port B address; **/b 2 = 1 8 F F** to preload the address containing the number of patterns in the cycle; and **/b 5 = 1 8 F D** to preload the delay time address. Press **init** and **go**. You should observe the following LED segments light in sequence.

Note: The digit select range is 00 through 07 and the segment select range (pattern) is 00 through 7F.

In this exercise, you have stored LED array and segment select numbers in memory, starting at hex. address 1900, to output a cyclic display. The patterns are on and off for an equal amount of time, and the amount of time is proportional to the number loaded at hex. addresses 18FD and 18FE. The number of patterns in the cycle is stored at hex. address 18FF. You can create your own display by changing the LED array and segment select numbers used in this program.

If you want to display alphanumerics, use the list of segment select numbers that appears after Exercise 10. For other patterns, simply derive your own segment select codes.

TTY PROGRAM EXERCISE

The following program exercise demonstrates the ability to key in a succession of characters (or message) on a TTY, store the resulting American Standard Code for Information Interchange (ASCII) codes in successive locations in the RAM of the MAC Tutor, and take the coded message stored in the MAC Tutor and print the individual characters out on the TTY. Several important computer-communications principles are illustrated in this exercise. They are:

- The use of a microprocessor to carry on 2-way communications with a peripheral device.
- The ability to store information and display it when desired.

- The facility of the TTY to convert numerical, alphabetical, and control characters to a standard format (ASCII code). (Texas Instruments' Silent 700 Electronic Data Terminal was used to develop this exercise.)

- The use of software to perform the serial-parallel conversion required to make TTY data compatible with data used by the MAC-8 microprocessor. (Serial data consist of a number of bits appearing on a single data line, spaced apart in time. Parallel data require separate lines for each bit, but all bits are available at the same time.)

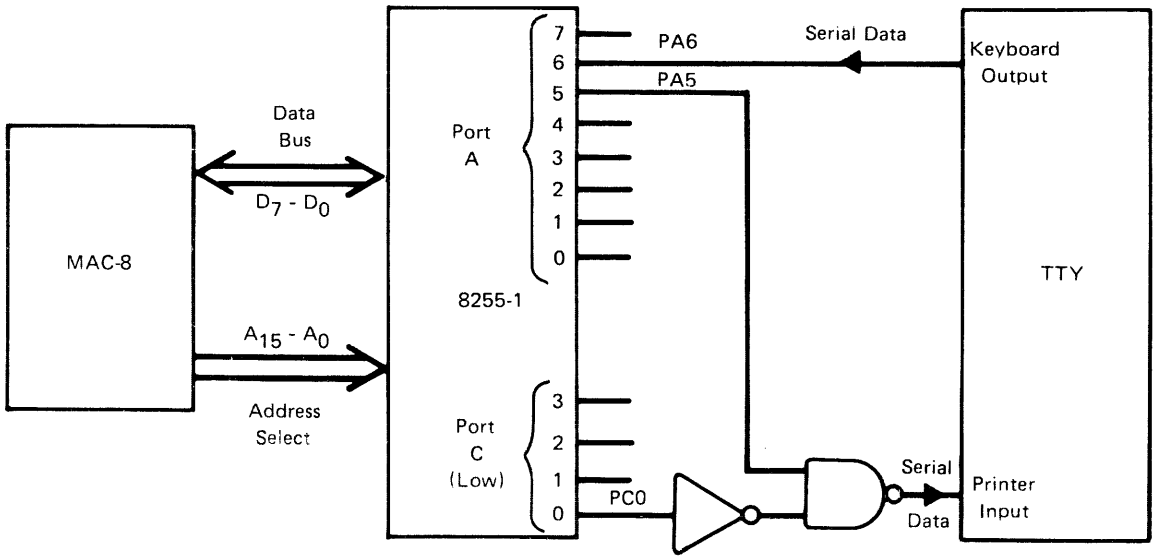
- The need to insert a controlled time delay into the program to reconcile the difference between the data rate of the TTY and the instruction execution time of the MAC-8 microprocessor.

BRIEF HARDWARE DESCRIPTION

The MAC Tutor provides two RS 232C connections for attaching TTY compatible terminals or modems capable of running at rates from 0 to 2400 baud. If a compatible TTY terminal is available and the proper interface connections are made, you can use the TTY keyboard and printer (under control of the MAC Tutor executive program) in place of the MAC Tutor keypad and display. This exercise assumes that the MAC Tutor-TTY terminal connections have been made and that communications will occur under control of a user program.

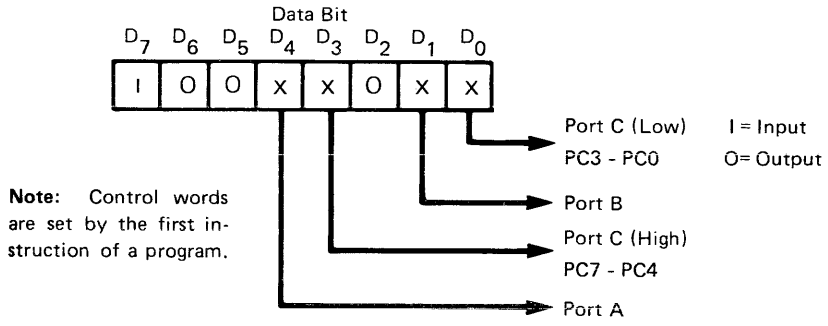
TTY Interface

The hardware interface between the MAC Tutor and TTY is shown in the following MAC Tutor-TTY Interface Diagram.



Simplified MAC Tutor-TTY Interface Diagram

Some of the details of the 8255 programmable peripheral interface chip were described in the I/O programming exercises. Recall that ports A and B of the 8255-1 chip and port A of the 8255-2 chip were used by the MAC Tutor keypad and display. The TTY interface uses only two bits of port A and the lowest bit of port C. An additional feature of the 8255 chip is its ability to control the direction of flow of the lower four bits of port C, independent of the high four bits. Although this feature is not important with respect to the TTY interface, it does affect the control word requirements of the 8255 chip. The control word format for the 8255 chip is shown below.



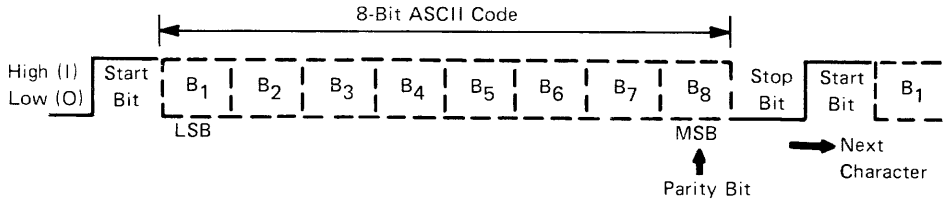
Control Word Format

To transmit serial data from the TTY keyboard to the MAC Tutor, port A is defined as an input port. Although not necessary, port B is defined as an input port for this mode of communication to keep the MAC Tutor display blank. If the TTY printer is to receive serial data from the MAC Tutor, ports A and C must be defined as output ports. The control word with respect to port direction is as follows.

Direction of Transmission	Control Word D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	Port Direction A	B	C (Low)	C (High)
TTY Keyboard to MAC Tutor	1 0 0 1 0 0 1 0	Input	Input	Output	Output
MAC Tutor to TTY Printer	1 0 0 0 0 0 1 0	Output	Input	Output	Output

Data Format

Referring to the MAC Tutor-TTY Interface Diagram, serial data from the TTY keyboard are received on PA6 of the 8255 chip and transmitted to the MAC-8 microprocessor on D_6. Conversely, serial data are transmitted by the MAC-8 microprocessor on D_5 and passed to PA5 of the 8255 chip. If the signal on PC0 is held low, the serial data are simply inverted and received by the TTY printer. The TTY printer issues serial data as shown below.



Serial Data from TTY Printer

The TTY issues 8-bit ASCII-coded characters. The first seven bits define the character and the eighth bit provides an even-parity check (i.e., the bit has a value that results in the 8-bit code containing an even number of logical 1 bits). Thus, a code of this length can uniquely identify 2^7 or 128 TTY characters. The following is a list of standard 7-bit ASCII codes, using hex. notation. The most significant bit is always 0, so the parity bit is not included in this code.

HEX. – ASCII CODE

00	NUL	20	SP	40	@	60	`
01	SOH	21	!	41	A	61	a
02	STX	22	"	42	B	62	b
03	ETX	23	#	43	C	63	c
04	EOT	24	$	44	D	64	d
05	ENQ	25	%	45	E	65	e
06	ACK	26	&	46	F	66	f
07	BEL	27	'	47	G	67	g
08	BS	28	(	48	H	68	h
09	HT	29	)	49	I	69	i
0A	LF	2A	*	4A	J	6A	j
0B	VT	2B	+	4B	K	6B	k
0C	FF	2C	,	4C	L	6C	l
0D	CR	2D	–	4D	M	6D	m
0E	SO	2E	.	4E	N	6E	n
0F	SI	2F	/	4F	O	6F	o
10	DLE	30	0	50	P	70	p
11	DC1 (X-ON)	31	1	51	Q	71	q
12	DC2 (TAPE)	32	2	52	R	72	r
13	DC3 (X-OFF)	33	3	53	S	73	s
14	DC4	34	4	54	T	74	t
15	NAK	35	5	55	U	75	u
16	SYN	36	6	56	V	76	v
17	ETB	37	7	57	W	77	w
18	CAN	38	8	58	X	78	x
19	EM	39	9	59	Y	79	y
1A	SUB	3A	:	5A	Z	7A	z
1B	ESC	3B	;	5B	[	7B	{
1C	FS	3C	<	5C	\	7C	|
1D	GS	3D	=	5D	]	7D	} (ALT MODE)
1E	RS	3E	>	5E	^ (↑)	7E	~
1F	US	3F	?	5F	_ (←)	7F	DEL (RUB OUT)

Notes:

1. When the parity bit is added, an uppercase B results in hex. 42, but uppercase C yields hex. C3 rather than 43 (as an example).

2. This ASCII code is generated by a mechanical TTY; the electronic terminals use a negative logical representation of the bits.

Some additional points about the format of the serial data are:

- Normally, the line connected to the TTY keyboard is at logical 0 (low). When a key is depressed (either alone or in conjunction with the shift or control key), a start bit (always at logical 1) is issued prior to the character code. Thus, the transition from low to high signifies the fact that a character code is to follow. Without this start bit, it is difficult to know when the code starts, particularly if the initial bits of the code are 0s.

- The data bits are issued, starting with the LSB and ending with the MSB, which is the parity bit. Actually, the data bits from the TTY keyboard are inverted when they enter the MAC-8 microprocessor because of the negative logic associated with the electronic terminal.

- A stop bit (always at logical 0) follows the MSB and enables recognition of the start of the next character. This level is maintained until the next TTY key is pressed. Some TTYs require two successive stop bits following the message.

- As the MAC Tutor-TTY Interface Diagram suggests, the TTY printer requires data in inverted form to that shown. Thus, the line to the printer is normally at a high level (logical 1) until the start bit appears as a high to low transition.

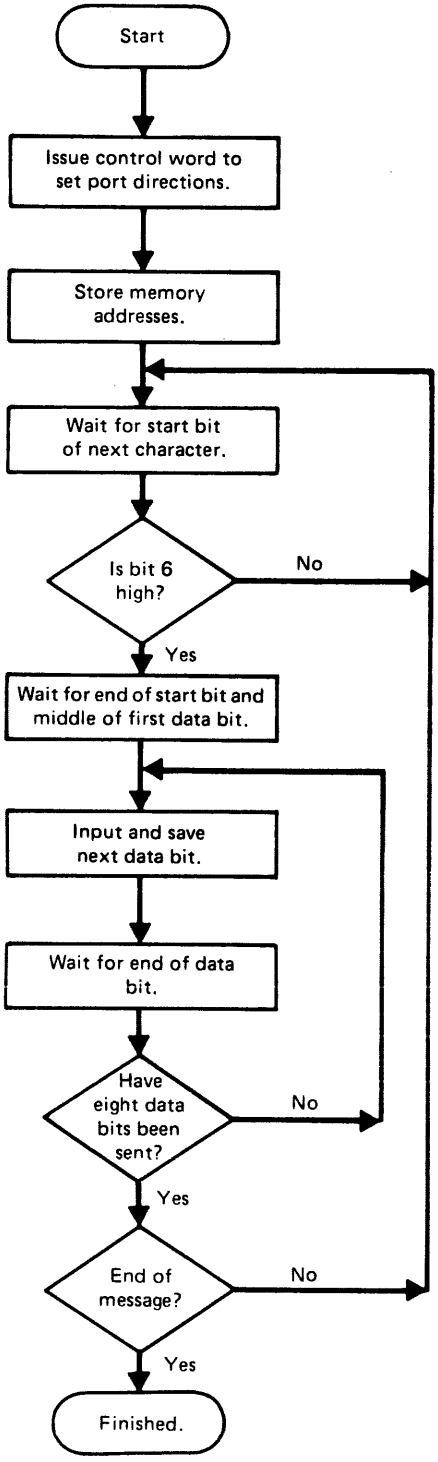
Exercise 14 — Store and Print Message

This program exercise is divided into two routines. The first routine takes the message keyed in on the TTY and stores it in the MAC-8 microprocessor RAM. The second routine prints the message out on the TTY. The two routines are described separately, with the complete program listing appearing at the end of the exercise.

Store Message Routine

The store message routine, shown in the flowchart, consists of the following segments:

- Initialization.
- Check for arrival of start bit.
- Input and save data bits.
- Check for end of message.
- Delay subroutine.



Store Message Flowchart

Exercise 14 (Continued)

Initialization

Instruction	Comments
*1F03=92;	Control word defines ports A and B as input ports.
b0=1F00;	Store port A address.
b1=1900;	Store starting address of message.
b5=18FE;	Store address of delay factor.
b8=18FC;	Store address of half-delay factor.

The incoming message is stored in successive memory addresses, starting with 1900. The preceding hex. addresses 18FC through 18FF store two 16-bit numbers, in conjunction with the delay subroutine, that reconcile the TTY and MAC-8 microprocessor timing differences. Registers b0, b5, and b8 can be preloaded with their addresses because they will not change when the program is executed.

Check for Arrival of Start Bit

Instruction	Comments
wait: if(!bit(6,*b0)) goto wait;	Loop until bit 6 is high.
pulse( );	Go to delay subroutine until end of start bit.
halfpuls( );	Go to subroutine that provides delay of half of bit width.

The program does not proceed until a TTY key is depressed, and the MAC-8 microprocessor idles until the TTY is ready to input data. This technique is called device polling and is commonly used in microprocessor applications.

Input and Save Data Bits

Instruction	Comments
a3=8;	Save number of data bits in ASCII code.

Exercise 14 (Continued)

Instruction	Comments
a4=0;	Save ASCII code in register a4.
loop: a7=*b0;	Input data bit.
a7=a7&40;	Mask all bits except bit 6.
a4=a4|a7;	Transfer to bit 6 position of register a4.
a4=a4>>>1;	Rotate data bit one position to right to make room for next data bit.
pulse( );	Go to delay subroutine until ready for next data bit.
--a3;	Repeat until all eight data bits are stored.
if(!zero)goto loop;	
a4=a4<<<1;	Because bits enter on line 6, rotate left twice so that
a4=a4<<<1;	LSB is bit 0, next to LSB is bit 1, etc.
*b1=a4;	Store character code in memory, outside of register space.
*b1++=~*b1++;	Complement character code and increment address for next character to be stored.

Note that the 8-bit ASCII code makes it convenient to perform the required serial-to-parallel conversion in an 8-bit a register (a4 in this case).

Check for End of Message

Instruction	Comments
if(!zero)goto wait;	Wait for next TTY character unless an ASCII code 0 is received.
return;	Return to executive program after message is completed.

A convenient way to terminate the message is to use the NUL control character on the TTY. The ASCII code contains all 0s and the routine exits from the loop when this character is received.

Delay Subroutines

Instruction	Comments
pulse: b6=*d5;	Fetch 16-bit delay factor stored in hex. addresses 18FE and 18FF.
delay: --b6;	Continue decrementing delay factor until it reaches 0.
if(!zero) goto delay;	
return;	Return to main program.
halfpuls: b9=*d8;	Fetch 16-bit half-delay factor stored in hex. addresses 18FC and 18FD.
halfdly: --b9;	Continue decrementing half-delay factor until it reaches 0.
if(!zero)goto halfdly;	
return;	Return to main program.

This simple type of delay subroutine has been used previously in the I/O programming exercises. In those exercises, the delay loop was used to change the LED display to a convenient rate for visual observance of the patterns. Changing the delay factor either speeded up or slowed down the display sequence, but did not alter the sequence. In this program, the delay factor must be chosen with care because the time during which the program is in the delay loop must correspond to the time width of a bit issued by the TTY. For this reason, and because timing delays are such an important aspect of microprocessor programming, a description of how the delay factor is chosen is included in this program exercise.

In this program, the TTY rate is set to 30 characters per second. This corresponds to a rate of 300 bits per second (called 300 baud) because each character consists of 10 bits (including a start bit and stop bit). Therefore, the time duration of a single bit is 1/300 second or 3.333 milliseconds (ms). The MAC-8 microprocessor, like all microprocessors, executes its instructions under the control of a clock. A clock is simply a crystal-controlled circuit that issues a repetitive series of high-low pulses at a particular rate (called the clock rate). Each MAC-8 instruction has a number of associated clock cycles that correspond to the total time required to fetch and execute the particular instruction. The clock cycle requirements for each instruction of the delay subroutine are as follows.

Exercise 14 (Continued)

Instruction	Clock Cycles
pulse: b6=*d5;	17
delay: --b6;	12
if(!zero)goto delay;	7
return;	8

The MAC Tutor operates at a clock rate of 1 MHz, so that the number of clock cycles corresponds directly to time in microseconds (μs).

Since the total delay required is several thousand μs (3.333 ms = 3333 μs), and each instruction takes approximately 10 μs to execute, several hundred instructions have to execute to allow the MAC-8 microprocessor to slow down to the rate of the TTY. This is the reason for storing a number in a register and repeatedly decrementing it. Since almost all of the time is spent in the delay loop, the time it takes to call the subroutine, initialize register b6, and return to the main program is relatively negligible (only about 30 to 40 μs). The delay loop itself consists of two instructions requiring a total of 19 μs. Thus, the number that must be stored initially in register b6 and preloaded into hex. addresses 18FE and 18FF is

$$b6 \cong \frac{3333}{19} = 175_{10} = 00AF_{16}$$

A second delay subroutine (called halfpuls) has been inserted into the program to improve the timing. An extra delay of half the bit width is inserted after the start bit is issued. This allows the MAC-8 microprocessor to sample each data bit near the middle of the pulse rather than at the beginning. This is good practice in an exercise of this type, as the system will be less sensitive to any drifting of the clock rate.

Since register b8 is used in this subroutine, it should contain the number

$$b8 = \frac{175_{10}}{2} = 87_{10} = 0057_{.16}$$

This number is preloaded into hex. addresses 18FC and 18FD.

Print Message Routine

The second routine of this program exercise, the print message routine, fetches a string of coded characters stored in memory and prints them out in succession on the TTY. Thus, when this routine is linked to the store message routine, the TTY message keyed in and stored in memory is printed out upon execution of this program segment. The print message routine is similar to the store message routine, so the same flowchart is applicable. The only differences are as follows: instead of serial data being input on PA6 of the 8255 chip, serial

data must be output on PA5 while keeping line PC0 low (see the MAC Tutor-TTY Interface Diagram). The program is now performing a parallel-to-serial rather than serial-to-parallel conversion. Also, bit-by-bit complementation of the ASCII code is not required in software since the hardware does this (note the NAND gate tied to the TTY printer). Finally, the program does not have to wait for a start bit for each character; it can issue the start bit since the characters are already stored.

The print message routine consists of the following segments:

- Initialization.
- Transmit start bit to TTY.
- Transmit data bits to TTY and check for end of message.
- Transmit stop bits to TTY and prepare to send new character.

Initialization

Instruction	Comments
*1F03=82;	Control word defines ports A and C as output ports.
*1F02=0;	Transmit low to PC0.
b1=1900;	Store starting address of message.

The message previously keyed in on the TTY is stored in ASCII code in memory, starting at hex. address 1900.

Transmit Start Bit to TTY

Instruction	Comments
next: *b0=0;	Transmit a low to PA5; signal goes to TTY printer (register b0 stores port A address).
pulse( );	Go to delay subroutine to establish required width of start pulse.

With reference to the description of timing, a low start bit of 3.333-ms duration is transmitted and inverted by hardware. The TTY then recognizes the succeeding transmitted bits as a character to be printed. The program returns to this point (next) after each character has been printed until a NUL character is encountered (signifies end of message).

Exercise 14 (Continued)

Transmit Data Bits to TTY and Check for End of Message

Instruction	Comments
a3=8;	Save number of data bits in ASCII code.
a4=*b1++;	Move ASCII code to register a4 and increment address for next character.
if(zero)return;	Return to executive program if end of message is encountered.
a4=a4>>>1;	Rotates LSB into bit 5 for subsequent transmission to TTY.
a4=a4>>>1;	
a4=a4>>>1;	
loop: *b0=a4;	Transmit bit on D_5 to TTY printer.
pulse( );	Delay to establish proper bit width.
a4=a4>>>1;	Rotates next bit to be transmitted to LSB.
--a3;	Repeat until all eight data bits are transmitted.
if(!zero)goto loop;	

It should be noted again that TTY serial transmission (in either direction) requires that the LSB be sent first and the MSB last.

Transmit Stop Bit to TTY and Prepare to Send New Character

Instruction	Comments
*b0=20;	Transmit a high to TTY printer.
pulse( );	Go to delay subroutine to establish required width of stop pulse.

Exercise 14 (Continued)

Instruction	Comments
goto next;	Return to point in program where start bit for next character is issued.

A high 3.333-ms stop bit is transmitted and inverted. This enables the TTY to recognize the start of the next character. Since the routine does not reach this point when a NUL is encountered, it always branches back to next.

Program Listing

The following is the complete program listing for the store message and print message routines.

Store Message

Instruction	Hex. Code	Starting Address
*1F03=92;	81 FF 03 1F 92	1800
b1=1900;	C0 1F 00 19	1805
wait: if(!bit 6,*b0))goto wait;	53 06 FE	1809
pulse( );	79 36 18	180C
halfpuls( );	79 3F 18	180F
a3=8;	80 3F 08	1812
a4=0;	20 40	1815
loop: a7=*b0;	85 70	1817
a7=a7&40;	98 7F 40	1819
a4=a4|a7;	90 47	181C
a4=a4>>>1;	34 4F	181E
pulse( );	79 36 18	1820
--a3;	28 38	1823
if(!zero)goto loop;	41 F1 17 18	1825
a4=a4<<<1;	34 41	1829
a4=a4<<<1;	34 41	182B
*b1=a4;	81 14	182D
*b1++=~*b1++;	2F 10	182F
if(!zero)goto wait;	41 F1 09 18	1831
return;	66	1835
pulse: b6=*d5;	C5 65	1836
delay: --b6;	68 68	1838
if(!zero)goto delay;	41 F1 38 18	183A
return;	66	183E
halfpuls: b9=*d8;	C5 98	183F
halfdly: --b9;	68 98	1841
if(!zero)goto halfdly;	41 F1 41 18	1843
return;	66	1847

Exercise 14 (Continued)

Print Message

Instruction	Hex. Code	Starting Address
*1F03=82;	81 FF 03 1F 82	1A00
*1F02=0;	21 F0 02 1F	1A05
b1=1900;	C0 1F 00 19	1A09
next: *b0=0;	21 00	1A0D
pulse( );	79 36 18	1A0F
a3=8;	80 3F 08	1A12
a4=*b1++;	87 41	1A15
if(zero)return;	64 01	1A17
a4=a4>>>1;	34 4F	1A19
a4=a4>>>1;	34 4F	1A1B
a4=a4>>>1;	34 4F	1A1D
loop: *b0=a4;	81 04	1A1F
pulse( );	79 36 18	1A21
a4=a4>>>1;	34 4F	1A24
--a3;	28 38	1A26
if(!zero)goto loop;	41 F1 1F 1A	1A28
*b0=20;	81 0F 20	1A2C
pulse( );	79 36 18	1A2F
goto next;	59 0D 1A	1A32

In addition to the store and print routines, the following data and addresses must be pre-loaded. *** 1 8 F C = 5 7 + 0 0 + A F + 0 0** stores half-delay factor 0057 in hex. addresses 18FC and 18FD and delay factor 00AF in hex. addresses **/b = 1 F 0 0, /b 5 = 1 8 F E, /b 8 = 1 8 F C** stores addresses in registers b0, b5, and b8.

To store and print a message, press **init** and **go** on the MAC Tutor. (The display will go blank.) Key in the message on the TTY. When you want to terminate the message, hold down **CTRL** and press **NUL**. The TTY will not print the message as it is being keyed in. Key in *** 1 A 0 0** and press **go**. The message will be printed out on the TTY and retained in memory.

The ASCII codes can be examined by pressing *** 1 9 0 0** and repeatedly incrementing the address. The program can be rerun for different messages by pressing **init** and **go** and repeating the above procedure. It is not necessary to reload any registers.

The final program exercises in this self-training manual are two follow-up exercises to the store and print message program exercises. They consist of:

- Varying the delay time factor that is preloaded into hex. addresses 18FE and 18FF. You will see that it is possible to increase or decrease it to some extent, but there are limits because of its relation to the TTY baud rate.

- Keying in a long message that requires more than one line to print out. You will find that a problem exists because a TTY carriage return and line feed will automatically occur when the end of the line is reached. Since the carriage return and line feed occur while the print message routine is being executed, the time required for this mechanical operation results in the loss of several characters from the message. You can overcome this problem in various ways, such as by using control characters in the message itself.