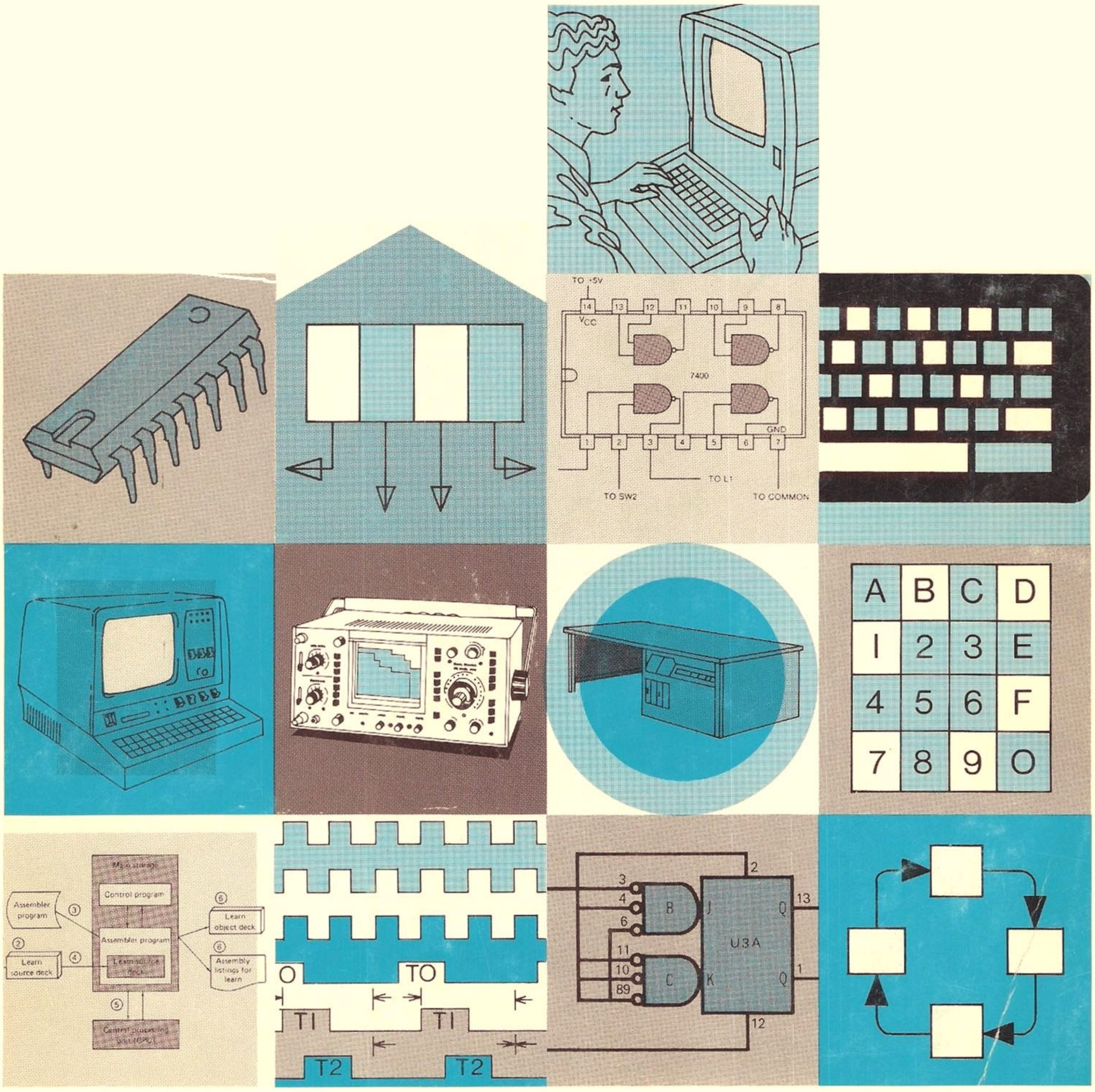
CONTROL DATA

Microprocessors: A Short Course



LAB MANUAL

Microprocessors: A Short Course

Lab Manual

CONTROL DATA

Pub. No. 76361376A

1/84

Table of Contents

(Blank Page)

Introduction to the Trainer Lab

Purpose

The purpose of this laboratory is to introduce you to the operational characteristics of the Mini-Micro Design Trainer (MMD-2) microprocessor trainer. You will also enter and execute several programs.

Objectives

- To identify functional sections of the MMD-2
- To enter and execute simple programs

Equipment

- MMD-2 Mini-Micro Design/Trainer

Introduction

The MMD-2 is a complete 8080-based microcomputer system, using a keyboard as the data entry method. This system provides you with an opportunity to program a typical microprocessor. For example, to enter a program using the MMD-2, you would enter (key in) the instruction code or data on the DATA INPUT keyboard, then key the STORE (STO) key on the FUNCTION INPUT keyboard. Depressing the STORE key places the information, keyed in on the DATA INPUT keyboard, into the read/write memory. The NEXT (NXT) key on the FUNCTION INPUT keyboard then selects the next sequential address into which information is to be written. After all the information has been stored and you are ready to execute the program, the starting address of the program is selected, and the program is run (executed) when the GO key on the FUNCTION INPUT keyboard is depressed. NOTE: This information will be covered in greater detail later in this laboratory.

Address and data information is displayed on the seven-segment readout displays. This information may also be displayed by the light-emitting diodes (LEDs) located on the left of the trainer. The LEDs will display the information in binary format. Remember, the binary format will have the least significant bit on the right of the LEDs. The LEDs are lit when a logical '1' is present on the line.

The MMD-2 has the capability to allow you to enter or read the displayed information in either the octal code or hexadecimal code. You also can switch between the two modes while entering or stepping through a program. The first key stroke after the MODE switch is changed will change the display readouts to the new mode.

Functional Areas

The following short descriptions will introduce you to the basic functional circuit areas of the MMD-2 trainer. Most of the major circuits will be covered throughout the course. You should look at each area of the MMD-2 as it is being described to become familiar with the types of components contained in that section.

Power

The Power switch is located in the left-rear area of the trainer next to the power fuse. When this switch is on, AC power is applied to the power supply. The power supply (not visible) is located inside the chassis and generates the required DC voltages for the circuits.

STD Interface

The STD Interface (STD should not be confused with the abbreviation for "standard") area contains the circuits to buffer and control the signals for the STD bus structure. The STD bus is the interface protocol used by the MMD-2 to connect various test and peripheral equipment.

CPU

This section contains the 8080 microprocessor and its required system controller (8228/8238). The system controller will monitor certain status conditions from the 8080 and select the correct read/write function. These functions include *memory read, memory write, I/O read,* and *I/O write.* The system controller also contains (within the chip) the data bus buffers. The CPU area also includes the address bus buffers.

Clock

This circuit establishes the operating timing of this microprocessor system. The crystal sets the operating frequency and the integrated circuits (ICs) generate the correct timing sequence.

PROM

This is the Erasable/Programmable Read Only Memory (EPROM) that contains the system operating program. The EPROM monitors the keyboards and mode select switches and allows either octal or hexadecimal entry.

The program contained in this EPROM was developed by E & L Instruments, Inc. and is called KEX for *K*eyboard *E*xecutive. It is this program that allows you to enter the data from the keyboards.

The remaining three locations in this section are spare and allow additional PROMs to be installed as needed. Thus, if additional programs are required or if you (the user) create new programs, written into PROM, the additional PROMs would be inserted here.

The PROM installed is really an μEPROM (Erasable/Programmable Read Only Memory). Note that the identification tag covers the center area of the EPROM. The reason for this is that the EPROM uses ultraviolet light to erase the program. Since almost all light contains this light frequency, the identification tag shades the enclosed circuits to prevent this light from degrading the system program.

RAM

These Random Access Memory (RAM) devices are the read/write memory devices that will contain the programs you generate in this and future labs. They are also known as volatile memory, because when you power the trainer off, the programs contained in these devices are erased (lost). The first RAM section (each section contains space foı two RAM chips), called System RAM, is two 256 x 4-bit RAMs that are a "scratch pad" for the system program and are not readily accessible to the user. The next section, called User RAM, consists of two 1k × 4-bit RAMs that will contain your programs. The last three sections are spares for future additions.

Serial I/O (not used)

This section would contain the circuits required to interface serial data communication devices, such as a cassette tape recorder (CASS). It also provides for serial-to-parallel conversion enabling the use of a video display system.

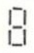
Control

This section contains the circuits for I/O and memory address decoders; that is, these circuits take the multiple address input lines and select a specific input port, output port, RAM, or PROM device.

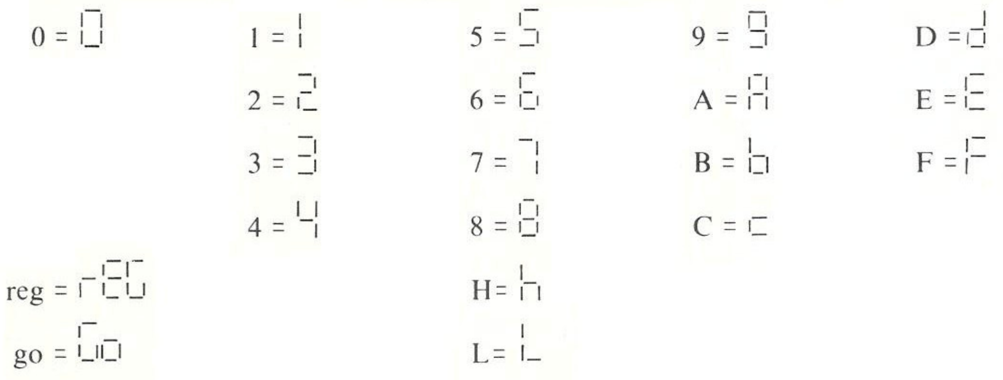
Keyboard and Display Interface

This section contains the circuits required to decode the data and function keys from the keyboard and the circuits to encode the data for the seven-segment displays. This section also contains these seven-segment displays.

The transistors located above the displays are the display drivers. These circuits supply the drive current required by the displays. The displays are called seven-segment because each section contains seven separate segments arranged in the following manner:



Each segment is independently driven and can "reproduce" each number and most alphabetic characters accurately. However, to avoid confusion between certain numbers and alpha characters, such as B ( ) and 8 ( ), certain "codings" of these characters are required. The following segment arrangements are used with the MMD-2. Note that the numerals are the same for both octal and hexadecimal. The "coding" interpretations are required for the alpha characters in the hexadecimal mode only.



Mode Selector

This Dual In-line Package (DIP) switch contains eight independent switches.

- Switch 6 Selects the octal or hexadecimal coding method for input and display.

  When this switch is in the OFF position, the OCTAL mode is selected. When this switch is in the ON position, the HEXADECIMAL mode is selected.

- Switch 8 Selects the output function of the PORTS 0, 1, and 2 lamp monitors. This function is either an output port or binary data display corresponding to the seven-segment display. This switch is labeled PORTS.

  When the PORTS switch is OFF, these ports (0, 1, and 2) are outputs of the microprocessor. When the PORTS switch is ON, the binary equivalent of the seven-segment display is shown.

- The remaining switches may perform other functions not related to this course. One of these functions is Write Protect for the RAM circuits. This means that if these switches are ON, you will NOT be able to enter data into the memory circuits.

NOTE: CHECK THESE SWITCH SETTINGS EACH TIME YOU START A NEW LAB. THE ONLY TWO SWITCHES YOU WILL NEED ARE 4 AND 8. ALL OTHERS SHOULD BE "OFF."

HALT, HOLD, ENABLE Indicators

These are lamp monitors for specific conditions of the 8080. These conditions are:

- HALT — When the program you write contains a HALT instruction and is executed, this lamp will light.

- HOLD - When the 8080 is in a hold mode waiting some input, this lamp will light.
- ENABLE - When the Interrupt circuits are enabled, this lamp will light. The interrupt circuits are not used in this course.

RESET

This switch resets the microprocessor. When this switch is depressed, the initial starting address for read/write memory is selected. The address is displayed and is 003 000_8 or 03 00_{16}.

You will enter and execute most of your programs starting at this address. Therefore, you will only have to use the RESET switch to get to this starting address whenever you start entry or execution of the program.

NOTE: Use caution when entering information using the FUNCTION keyboard so that the RESET switch is not accidentally depressed, causing the program to reset.

Keyboard

The keyboard keys specify the data or function to be entered into the CPU. The following text lists the keyboard keys that are available and explains their functions.

Function Input	
PRE (PREVIOUS)	This is a decrement address function and selects the memory address one position back from the address being displayed each time the key is depressed. The new address and data are displayed.
STO (STORE)	Writes (stores) the data entered into the RAM memory.
NXT (NEXT)	This is an increment address function and selects the next sequential memory address to either write or read data. The new address and data are displayed.
STP (STEP)	Steps through each program instruction. Each time this switch is keystroked, the instruction is executed. This allows you to execute an entire program instruction-by-instruction.
HI (HIGH)	Selects the H register of the 8080 to enter the high-order 8 bits of the 16-bit address.
LO (LOW)	Selects the L register of the 8080 to enter the low-order 8 bits of the 16 bit address.
GO	Initiates the execution of the program.
MEM (MEMORY)	Data will be written into or read from the memory at the address specified (displayed).

REG (REGISTER)	Data will be written into or read from the specified (displayed) register.
CAN (CANCEL)	This key is basically a "clear entry," which returns the displayed data back to the previous data if the STORE key has not been depressed.

Data Input

The DATA INPUT keyboard allows you to enter the desired address and data.

When the MODE SELECT switch (the eight position DIP switch on the right side) is in the OCTAL mode (OFF), the DATA INPUT keys 0 through 7 are usable. The remaining keys (8 through F) will have no effect.

When the MODE SELECT switch (the eight position DIP switch on the right side) is in the HEXADECIMAL mode (ON), the DATA INPUT keys 0 through F are usable.

Input Port

This is an external device input for address zero. The input bus to this chip is not used during any of these labs.

Port 1, Port 0, Port 2

These are the output ports containing lamp monitors. When the PORTS—MODE SELECT switch is in the binary (ON) position, these lamp monitors indicate the binary code for the numbers being displayed in the seven-segment display. Following is the format for this output:

- Port 1 indicates the upper 8 bits of the memory address.
- Port 0 indicates the lower 8 bits of the memory address.
- Port 2 indicates the 8-bit data word contained in the memory address.

When the PORTS—MODE SELECT switch is in the port (OFF) position, the lamps in the selected output address will be enabled. Only the data being output to a specific address (port) will be displayed.

Remember, the least significant bit (2^0) is on the right, and the most significant bit (2^7) is on the left. When all indicators are lit, the codes are 377_8 or FF_{16}.

PROM Programmer (not used)

This area enables you to copy programs from the MMD-2 memory into a new PROM. Thus, if you had written a program into RAM and wanted to store it permanently, you could use this area to "burn" (program) a PROM device.

This completes the functional description of the MMD-2 at this time. You will now use the MMD-2 and apply what you have just learned to enter and execute several simple programs (routines).

Procedure

Step 1. Apply AC power to the trainer by plugging the power cord into a convenient AC outlet.

Step 2. Set the MODE SELECT switches as follows:

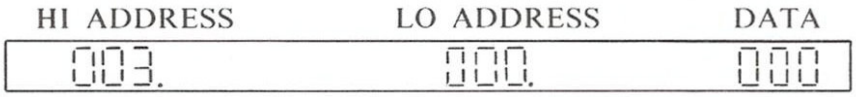
HEX/OCTAL to octal (switch 6 OFF)

PORTS to binary (switch 8 ON)

ALL OTHER MODE SWITCHES OFF

Step 3. Apply DC power to the circuits by switching the power switch located on the rear of of the unit. NOTE: When power is on, this switch will glow red.

The seven-segment displays are arranged in the following order:



When power-on occurs, the PROM program will set the starting address of a specific RAM location. This address will be 003 000$_8$, as displayed in the high and low address sections of the display. The LSB position of each of these sections will also contain a decimal point (period).

Question. Does the high address equal 003? ____________________

Question. Does the low address equal 000? ____________________

(The answer should be yes to both of these questions.)

The data section will display the data contained in memory location 003 000$_8$. This data will be randomly selected on power-up, because the memory (RAM) is volatile.

Step 4. Depress the **MEM** key on the FUNCTION keypad, then the **0** key on the DATA INPUT keypad three times.

The MEM key verifies or selects the MEMORY mode of operation. This enables you to read/write data in the read/write (RAM) memory and not in a register.

Question. Does the display now show 003.000.000? ______________________ (yes)

Look at the lamp monitor (Ports 0, 1, and 2). These are shown as below:

Port 1	OO OOO O●●	High Address LEDs
Port 0	OO OOO OOO	Low Address LEDs
Port 2	OO OOO OOO	Data LEDs

The two least significant bits in Port 1 should be lit, and all others should be off. If not, check the PORTS switch on the MODE SELECT switch.

Step 5. Depress the **3** key on the DATA INPUT keypad.

Question. What is the number shown in the DATA display? ______________________

Question. What is the octal code displayed in Port 2? ______________________

(Both should show 003 displayed.)

Step 6. Depress the 2 key.

Step 7. Depress the 7 key.

Question. What is the number shown in the data display? ______________________

Question. What is the octal code displayed in Port 2? ______________________

(The display should read 327 and the octal code in Port 2 should be as shown by the filled circles.)

Port 1	O	O	O	O	O	O	●	●	(LSB)
Port 0	O	O	O	O	O	O	O	O	
Port 2	●	●	O	●	O	●	●	●	

You have now selected 1 byte of data to be entered. Remember, when entering a code on the MMD-2, the most significant bit is entered first.

Step 8. Depress the **4** key.

Step 9. Depress the **1** key.

Step 10. Depress the **3** key.

Question. What is the code displayed either on the seven-segment display or Port 2?

____

Question. If 413_8 was the number, why is the code displayed not equal to 413_8?

____

(Octal codes are 3 bits each; therefore, to show an octal code for numbers between 400 and 777 requires 9 bits. Since this is an 8-bit device, the ninth, or most significant bit, is lost. You cannot enter octal codes for numbers between 400 and 777.)

Address Selection

To set the starting address of a program (read or write), you must first enter the upper byte, then the lower byte. The R/W memories in which you can write data have addresses from $000\ 000_8$ through $003\ 377_8$.

To select address $002\ 000_8$:

Step 11. Depress **0** on the DATA keypad.

Step 12. Depress **0** .

Step 13. Depress **2** .

Step 14. Depress **HI** on the FUNCTION keypad.

Step 15. Depress **0** .

Step 16. Depress **0** .

Step 17. Depress **0** .

Step 18. Depress **LO** on the FUNCTION keypad.

When the **HI** was depressed, the octal code (002) was stored in a register within the 8080 microprocessor called the H (for HIGH) register. This octal code was also placed on the output lines in Port 1 and the left seven-segment display. When the **LO** was depressed, the octal code (000) was stored in a register within the 8080 microprocessor called L (for LOW) register. This octal code also was placed on the output lines in Port 0 and the center seven-segment display.

Question. When you depressed either the **HI** or **LO** , did the code displayed in Port 2 change or DATA (right) seven-segment display change? ________________________

(Either yes or no would be a correct answer. When HI or LO was depressed, the information stored in the R/W memory at that location was displayed in Port 2 and the DATA display.)

Step 19. Depress the RESET switch.

Question. What happened to the HIGH and LOW address displays? ________________

____

(They changed from 002 000$_8$ to 003 000. This is the program-controlled starting address for the RAM memory.)

Step 20. Depress the **NXT** key on the FUNCTION keypad three times. Each time you depress this key, notice what happens to the LOW address display and Port 0.

Question. Describe this change. ________________

____

(Each time the **NXT** key is depressed, the LOW address counted up (incremented) by 1.)

Step 21. Depress the **PRE** (Previous) key on the FUNCTION keypad three times. Each time you depress this key, notice what happens to the LOW address display and Port 0.

Question. Describe this change. ________________

____

(Each time the **PRE** key is depressed, the LOW address counts down (decrements) by 1.)

You should keep the function of these two keys in mind as they will become very helpful when you have to verify a program.

Enter a Program

So far you have learned how to enter information using the DATA keyboard and how to select another memory address using the DATA keyboard and the HI and LO keys on the FUNCTION keyboard. You also learned how to increment and decrement the memory address using the NXT and PRE keys. Finally, you saw that depressing the RESET switch caused the RAM starting address to become 003 000$_8$. Now, you will learn how to enter a short program into the read/write (RAM) memory.

Instead of seeing a single key and the darker print in each step, you will now see the data byte and function keys as a complete step. Simply depress the keys as they are presented.

Step 22. PORTS—MODE SELECT switch to port (switch 8 OFF).

Step 23. RESET

You have now set the lamp monitors to indicate an output device. All lamps should be out in Ports 0, 1, and 2. You have also selected the starting address of 003 000$_8$ when RESET was depressed.

Step 24. 0 7 6 STO.

When you depress the STO key, you place the data (076) code into the RAM memory at location 003 000$_8$.

Question. How would you increment to the next memory location?________________

____(See Step 25.)

Step 25. NXT.

Now we will continue entering this program.

Step 26.	0	0	5	STO	NXT.
Step 27.	3	2	3	STO	NXT.
Step 28.	0	0	0	STO	NXT.
Step 29.	1	6	6	STO	

Question. Do the displays show 003 004 166? ________________________

(If not, you have an error in program entry. Repeat steps 23 through 29.)

Verify a Program

It is good practice to restep through a newly entered program and verify that the data entered is correct. To verify the program you just entered:

Step 30. RESET.

Question. Does the DATA display equal 076? ________________________

Step 31. NXT.

Question. Does the DATA display equal 005? ________________________

Step 32. NXT.

Question. Does the DATA display equal 323? ________________________

Step 33. NXT.

Question. Does the DATA display equal 000? ________________________

Step 34. NXT.

Question. Does the DATA display equal 166? ________________________

(If your answer to any of the above questions is no, you have a code entry error.)

To correct a code entry error, you have several options available to get to the address that contains the error.

These include:

- Depressing the NXT key to step up to the incorrect data address.
- Depressing the PRE key to step back to the incorrect data address.
- Selecting the H and L address code of the incorrect address.

NOTE: If the address to be selected has the same high-order address byte (i.e., 003) then you need only to select the low-order address byte.

For example, change the code at address 003 003 to read 001 instead of 000. (You should show address 003 004_8 presently.)

Step 35. 0 0 3 L.

You have now selected address 003 003_8, and the data should show 000.

Step 36. 001 STO

You have now replaced the data (000) contained in location 003 003_8 with 001.

Execute the Program

You will now execute the program you have just entered and verified.

This program is a simple routine that will load the number 5 into the accumulator register (A register) and output that number to output Port 1. Then stop.

Question. What is the starting address of this program? ________________________

____(003 000)

Question. The initial or starting address is 003 000_8. The easiest way to get to the initial or starting address would be to key the ______ switch. Two other methods would be to key ______ LO or depress the ______ key.

(RESET, 000, PRE)

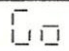
Step 37. Key in RESET GO.

Question. Describe what happened. ____

____

____

____

(The DATA display displayed  (GO) and output Port 1 showed the binary code for a 5.)

Modify the Program

If you were told to modify the program so that the data displayed in Port 1 were 7 instead of 5, several questions would immediately be asked. These questions could be "What is the data instruction?" and "Where is it located?"

Now, you should start to see the importance of complete and accurate program documentation. A proper program listing will identify all necessary information. The program listing for your program is:

Address	Code	Mnemonic	Comment
003 000	076	**MVI A**	Move the immediately following data
003 001	005	(data)	to the A register.
003 002	323	**OUT**	Output the data to
003 003	001	(port address)	Port 1.
003 004	166	**HLT**	Halt.

Question. What is the address of the data (contents of the A register) to be output?________

(The data is located at address 003 001.)

Since the program was started when the GO key was depressed, the microprocessor is in a "run" condition even though a HALT command was executed. Although the microprocessor has stopped execution of instructions, it is still in a GO condition. Before you can modify any data, you must reset the 8080.

Question. How do you stop (reset) the program?________________________________

(The RESET key)

Now modify the contents of location 003 001 to change the data from 005 to 007.

Step 38. Select address 003 001 (after RESET key depressed).

Step 39. Key in 007 STO, RESET, GO.

Question. Did the Port 1 lamps change from binary 005 to binary 007?____ If not, stop the computer and verify the program as shown.

Address	Data (Code)
003000	076
001	007
002	323
003	001
004	166

Final Program

You will now enter another program. It will also require you to enter the main program at one address and a subrouting program starting at another address.

The program will operate as follows:

The accumulator register in the 8080 will be incremented by one. It will then output to all output ports; at this time, it will jump to a time delay subroutine. The time delay will be specified as 10 milliseconds (msec) in the subroutine. After 10 msec, the main program will be jumped back to. The main program will now jump to the starting address and the sequence will repeat.

(Do not enter the program at this time. You should become familiar with the codes and comments. You will enter it later.)

The program listing is:

MAIN PROGRAM

Address	Code	Name	Comment
003 000	074	**INR A**	Increase A by 1.
003 001	323	**OUT**	Output
003 002	002		to Port 2.
003 003	323	**OUT**	Output
003 004	000		to Port 0.
003 005	323	**OUT**	Output
003 006	001		to Port 1.
003 007	315	**CALL**	Get the subroutine at location
003 010	030		low-order address byte,
003 011	003		high-order address byte.
003 012	303	**JMP**	Jump to location
003 013	000		low-order address byte,
003 014	003		high-order address byte.

TIME DELAY SUBROUTINE

Address	Code	Name	Comment
003 030	365	**PUSH PSW**	Store the value of the stack pointer (SP) register pair.
003 031	325	**PUSH D**	Store the value of the D-E register pair.
003 032	021	**LXI D**	Enter the following data
003 033	046		into E (LOW-order byte).
003 034	001		into D (HIGH-order byte).
003 035	033	**DCX D**	Decrement D-E by 1.
003 036	172	**MOV A,D**	Move register D to A.
003 037	263	**ORA**	OR registers E and A.
003 040	302	**JNZ**	If not equal to 0 go back
003 041	035		to LOW-order address byte,
003 042	003		HIGH-order address byte.
003 043	321	**POP**	Return the D-E value from memory.
003 044	361	**POP**	Return the SP value from memory.
003 045	311	**RET**	Return to the main program.

The time delay subroutine loads the value 001046_8 into the D-E registers and subtracts one. It then moves the D register data byte and ORs the bytes (D and E). If not equal to 0, it returns to the decrement instruction.

If equal to 0, it restores the address and contents of the main program, then returns control to the main program. The main program then starts the entire process over again.

Question. What is the starting address of the main program? ______________________

(003 000)

Question. What is the starting address of the time delay subroutine? ________________

(003 030)

You will now enter and execute the program. We will also delete the function key notation. This will approximate a standard program load. Don't forget the function of the keys RESET (stop/reset), HI (high-address byte), LO (low-address byte), STO, NXT (step/next), GO.

Step 40. Select main program starting address 003 000.

Step 41. Enter and store the following information:

```
074
323
002
323
000
323
001
315
030
003
303
000
003
```

Step 42. Select the subroutine starting address 003 030.

Step 43. Enter and store the following data:

```
365
325
021
046
001
033
172
263
302
035
003
321
361
311
```

Step 44. Select main program starting address and start (execute) the program 003 000.

Question. Are the output port LEDs incrementing? ______________________________

(If they are not, there is an entry error; verify your program.)

Step 45. To increase the delay time of the LEDs, modify the contents of address 003 034_8 to be 040_8.

Step 46. To decrease the delay time of the LEDs, modify the contents of address 003 034_8 to be 000_8.

After you have changed the delay times of the program and verified the various lamp sequences, depress the RESET key to stop the program.

Step And Register Functions

The STP and REG FUNCTION keys will play an important role when you are attempting to debug a program. They will also help you understand the operation of this 8080 microprocessor. The STP key allows you to step through a program instruction-by-instruction. That is, when the STP key is depressed, the instruction (displayed) will execute and stop at the next instruction. For example, without changing the program for incrementing the output port LEDs you have just completed, do the following:

Step 47. RESET STP. The address displayed should read 003 001, and the data field should read 323.

Step 48. STP. The address displayed should be 003 003, the data field should read 323, and Port 2 should contain random data (this is because it is not known what data is initially in the A-register).

Step 49. Depress STP two more times. You should now have 003 007 315 displayed.

This is the **CALL** instruction. Its function is to jump to the Time Delay Subroutine located at address 003 030_8. Therefore, when you STP again, the address and data displays should show the address and first instruction of this subroutine.

Step 50. STP.

Question. Did the program execute correctly? ______________________________ (yes)

Step 51. Load the following program into memory starting at address 003 020:

```
076     MVI A     Move immediate to A
010               (data to A).
075     DCR A     Decrement A by 1.
303     JMP       Jump to location
022               (022 L),
003               (003 H).
```

This program will load a number (10_8) into the A register, decrement the A register by one and jump back to the decrement instruction. The program will continue in this loop until *you* stop it.

The REG key allows you to see the contents of the various registers of the 8080 microprocessor. The register selection will occur when the REG key is depressed.

Step 52. Select address location 003 020. (Remember, the RESET key will select address 003 000 only.) Do not press GO at this time.

Step 53. Depress REG.

The display should now read rEG A (data). Each time you depress the NXT key, the next successive register and its contents will be displayed. The order of these registers is A,B,C,D,E,H,L,SH,SL,FL(repeat). The SH and SL registers are the high and low bytes of the stack pointer register and the FL will display the flag register (covered later).

Step 54. Depress NXT to verify the order of registers to be displayed. When you reach the A register again, stop.

Step 55. Depress STP. The A register should now show 010 as the data.

Step 56. STP. As you depress STP, the A register was decremented by one.

Question. What are the contents of the A register now? ____________________ (007)

(Remember, you are in the octal mode. 010_8 is equal to 8_{10} and 8_{10}-1_{10}=7. Therefore, the A register displays 007.)

Step 57. Continue to depress the STP key. Notice that the A register will change from 000 to 377 and will continue to count down. The program would continue to run in this loop until you stopped it because the system is not looking for any condition such as zero or negative sign. NOTE: The negative sign would occur at the 000 to 377 step and is a status flag set by the results of the operation performed. This is done by the ALU (arithmetic logic unit) in the 8080.

You may continue to experiment with any portion of this lab that you wish to. For example, you could step through the time delay subroutine while observing the registers, or you could continue to vary the time delay counts.

Step 58. Depress the RESET key and turn off the power to the MMD-2.

Summary

In this lab, you were introduced to the various functional areas of the MMD-2 Mini-Micro Design/Trainer. You learned the functions of the PRE, STO, NXT, STP, MEM, REG, HI, LO, and RESET keys, as well as the data input keys. You entered, stored, verified, modified, and executed several programs. These programs used Ports 0, 1, and 2 as output ports. You also used these same ports as a binary indication of the address and data display. One of the programs you entered contained a time delay routine. With this routine you selected various parameters for the time delay and saw the effect these had on the output. Finally, you used the STP key to step through a program. This enabled you to see what happened during the execution of a program. You also selected the REG key and looked at the contents of various registers, as well as watched the A register decrement through zero and continue.

Introduction To Machine Language Lab

Purpose

The purpose of this laboratory is to provide you with practice in converting simple programs from one base number system into another base number system. You will also enter and execute these programs in both the octal and hexadecimal mode.

Objectives

- To convert simple programs written in one number system into octal and hexadecimal codes
- To enter a program in both octal and hexadecimal and verify results of program operation
- To convert results of program operation into octal, binary, decimal, or hexadecimal code

Equipment

- MMD-2 Mini-Micro Design/Trainer

Introduction

Most 8-bit microprocessors list or identify their machine language instruction sets as a string of ones and zeros. This string will be 1 byte (8 bits) in length and will be the specific code for a specific operation or function (called instructions). A series of specific instructions and data in the correct sequence is a program.

For example, this string of bits may appear as follows: 10100101. If you had to enter a program containing 100 or more instructions by entering each bit separately, you can see the probability of error is high. To lessen the entry error, many microprocessor systems use a keyboard entry method. This keyboard may use either the octal or hexadecimal coding scheme. These codes are encoded by an electronic circuit called an *encoder*. The encoder then places the corresponding bits on the input bus to the microprocessor.

What you have to do, then, is to determine the octal or hexadecimal equivalent code for the given bit string. You may also have to convert a program from one code to another if the entry method has changed.

Let's look at the bit string 10100101 again. This code would be 245 octal (10 100 101 = 245), or A5 hexadecimal (1010 0101 = A5). Thus, if the keyboard entry method were octal, you would have to key in 2, then 4, then 5. As you can see, this is much easier than having to key in 10100101, and hexadecimal is easier yet.

Probably the easiest way to convert from octal to hexadecimal (or vice versa) is to convert to binary code first, then into the required code. You should already be familiar with these code conversions.

Procedure

Step 1. Convert the following binary coded program into octal and hexadecimal codes.

	Binary	Octal	Hex.
1	00111110	____________	____________
2	01011011	____________	____________
3	11010011	____________	____________
4	00000000	____________	____________
5	01110110	____________	____________

Step 2. Plug in the MMD-2 to a convenient AC outlet.

Step 3. Turn the power switch on.

Step 4. Verify the MMD-2 is in the memory function by depressing the MEM key.

Step 5. Place the MMD-2 in the octal entry mode by placing the HEX/OCTAL MODE SELECT switch in the octal (switch OFF) position.

Step 6. You will now enter the octal code program from step 1, starting at location 002000_8.

Key in	002	HI	
	000	LO	
Octal codes	—	STO	NXT
from step 1	—	STO	NXT
in order	—	STO	NXT
	—	STO	NXT
	—	STO	
	000	LO	

Step 7. Verify MMD-2 is in the port mode of operation by placing the PORTS—MODE SELECT switch in the OFF position.

Step 8. Execute the program you just entered by depressing GO.

The purpose of this program is to move a byte of data to the accumulator and to output that data byte to Port 0, then stop. The program listing is as follows:

00111110	is move the following data to the accumulator
01011011	is the data
11010011	is an output instruction
00000000	is the output port
01110110	is halt

Question. Does output Port 0 equal 01011011? ______________________

(If it does not, you may have made a code conversion or entry error. Compare your codes with the following: 076, 133, 323, 000, 166. If they compare, verify the entered program by stopping the MMD-2. Enter RESET 002 HI 000 LO to start, then NXT, comparing the data display with your program. When the program is operational, continue on with the next step.)

The following program will add two numbers together and output the result to Port 2.

Step 9. Convert the following program into hexadecimal (if required) and record your answers in the spaces provided following this listing.

1	076_8	Move the following data into the A register.
2	01001001_2	Data.
3	016_8	Move the following data into the C register.
4	49_{16}	Data.
5	10000001_2	Add the contents of C and A registers.
6	323_8	Output
7	02_{16}	to Port 2.
8	01110110	Halt.

1	______________	Move the following data into the A register.
2	______________	Data.
3	______________	Move the following data into the C register.
4	______________	Data.
5	______________	Add the contents of C and A registers.
6	______________	Output
7	______________	Port 2.
8	______________	Halt.

Step 10. Place MODE SELECT to hexadecimal mode (HEX/OCTAL switch ON).

Step 11. Enter and execute the program, starting at location 0300_{16}.

Question. What are the binary contents of Port 2 when the program stops? ______________________ (10010010_2)

Question. Convert the contents of Port 2 into octal. ______________________ (222_8)

Question. Convert the contents of Port 2 into hexadecimal. ______________________ (92_{16})

Question. Convert the contents of Port 2 into decimal. ______________________ (146_{10})

Question. Which two positions (address locations) in the program contain the ______ data to be added together? ______________________ (0301, 0303)

NOTE: If the program did not stop or your answers do not agree, you have either a conversion or data entry error. Verify your program with the following in order: 3E, 49, 0E, 49, 81, D3, 02, 76.

The following program will subtract 4 from 10 and place the results (6) in output Port 2. To complete this program you will have to locate the mnemonic for the instruction in the instruction set and convert the given code into octal or hexadecimal. The instruction set (mnemonics and codes) is found in the appendix of this manual.

Step 10. Code the following program (in either octal or hexadecimal).

Code	Mnemonic	Comments
______	**MVI r, data**	DDD equals 111.
______	(data)	The decimal number 10.
______	**MVI r, data**	DDD equals 001.
______	(data)	The decimal number 4.
______	**SUB r**	SSS equals 001.
______	**OUT port**	——
______	(port number)	Output Port 2.
______	**HLT**	Stop.

The **MVI** instruction is located in the group called DATA TRANSFER and the **SUB** is in the ARITHMETIC group. As you find each of these instructions, notice the similarity between the instructions within the same group. In many cases it is only the difference of one bit between two different types of operation.

As you can imagine, this can be the difference between a program's operating properly or not.

Step 11. Load and execute the program you just coded. Start at location 003000_8 or 0300_{16}. Don't forget to set the MODE SELECT switch to the base system (octal/hex) you are using.

Question. What are the contents of Port 2 when the program stopped?

____ (006_8 or 06_{16})

If your answer does not agree, you have either a coding or entry error. Compare your codes with the following: octal = 076, 012, 016, 004, 221, 323, 002, 166 and hexadecimal = 3E, 0A, 0E, 04, 91, D3, 02, 76, then verify/modify the program entry.

Step 12. RESET the trainer and turn off the power.

Summary

In this lab, you learned that it is very necessary to be able to convert from one base number system into another. This is because of the various possible entry methods used by computer/microprocessor manufacturers. You practiced converting from several different number systems into octal or hexadecimal so you could enter and execute programs on MMD-2 trainer. Then you practiced converting the results back into the original number system to verify proper operation.

You also learned how to code a simple program when given the mnemonics for the 8080 microprocessor. This entailed converting the binary digits and two decimal numbers into octal or hexadecimal code for entry. You may have also learned that it is very easy to make a conversion or entry error.

Addressing Techniques Lab

Purpose

The purpose of this laboratory is to demonstrate the various addressing modes of the 8080 microprocessor. These modes include *direct, indirect, immediate,* and *register*.

Objectives

- To enter the various 8080 addressing modes
- To execute the various 8080 addressing modes

Equipment

- MMD-2 Mini-Micro Design/Trainer

Introduction

The 8080 is capable of addressing data stored in a register or memory by using any one of four methods. These methods are called addressing techniques or addressing modes. As mentioned above, these modes are:

- Direct mode - the following 2 bytes state the exact memory address.
- Indirect mode - the address is contained in a register pair, and the register pair is specified in the instruction.
- Immediate mode - the following 1 or 2 bytes of the instruction contain the data.
- Register mode - the instruction specifies a register or register pair in which data is located.

Programs often contain branch or jump instructions that direct program operation to be performed from some other address. It is these jump instructions that can cause a program to "loop" (repeat itself) until either certain conditions are met or to run indefinitely. These branch instructions specify the address by either the direct addressing mode or the indirect addressing modes.

Procedure

Direct Addressing

The **LDA addr** and the **STA addr** are two examples of a direct addressing mode. The **LDA** will load the accumulator with the data contained in the memory address specified by the 2 bytes of data following the **LDA** op code. The **STA** will store the contents of the accumulator into memory at the address specified by the 2 bytes of data following the **STA** op code.

Step 1. Set the HEX/OCTAL MODE SELECT switch to OCTAL and PORTS switch to OFF.

Step 2. Apply power to the MMD-2.

You may use the HEX mode if you want. However, you will have to recode the programs into the hexadecimal code.

Step 3. Select address 002000_8. This will be the location from which the data will be taken by the **LDA** instruction.

Step 4. Enter 333 and store this data byte.

Step 5. Select address 003000_8. This will be the program start location.

Step 6. Enter and store the following program.

Location	Code	Mnemonic	Comment
003000	072 (START)	**LDA addr**	Load A with the
001	000		contents of this
002	002		location.
003	323	**OUT**	Output
004	002		to Port 2.
005	166	**HLT**	Stop.

Step 7. Execute the program at START (location 003000).

Question. What are the contents of output Port 2?____________________(333)

Question. Describe the operation of this program.

____

____

____

____

____

(This program loaded the accumulator with the contents of 002000_8, which was 333. Then it output the contents of the accumulator to Port 2 and stopped. Port 2 now displayed 333_8.)

Step 8. Starting at location 003003₈, enter the following:

Location	Code	Mnemonic	Comment
003003	062	**STA addr**	Store A at the
004	030		address specified
005	003		by these two locations.
006	166	**HLT**	Stop.

This program modification will now store the contents of the accumulator in memory at location 003030₈.

Step 9. Execute the program at START (003000₈).

Step 10. Select address 003030₈. (Don't forget to RESET the trainer before attempting to select the address.)

Question. What are the contents of address 003030₈ as displayed by the seven-segment data display?____ (333)

As you can see, both the **LDA** and **STA** instructions referenced a storage location directly by the 2 bytes following the instruction itself.

Question. Since the storage address is 16 bits (2 bytes) in length, which byte appears first in the program (low or high)? ________________________________ (low)

Indirect Addressing

The **LDAX rp** and **STAX rp** are examples of an indirect memory addressing technique or mode. In both of these instructions, bits 4 and 5 select which register pair contains the address. Only two register pairs may be used, either B-C or D-E. (To find out what the bit pattern is, see the introduction section of the 8080 Instruction Set found in the appendix of this manual. Look under "Symbols and Abbreviations" for RP.) When specifying a register pair, the low-order address byte must be in C or E and the high-order address byte in B or D.

For this example we will continue to use address 002000₈ for the data contents and address 003030₈ to store that data.

The first thing the program must do is to load each register pair with the address bytes; then it selects that pair, loads the data from memory, transfers the data from the register pair to the A reg, then stores the data in memory specified by the **STAX rp** instruction.

Step 11. Enter the following program, beginning at location 003000_8.

Location	Code	Mnemonic	Comments
003000	001(START)	**LXI B**	Load register pair
001	000		(C data),
002	002		(B data).
003	021	**LXI D**	Load register pair
004	030		(E data),
005	003		(D data).
006	012	**LDAX A**	Load A from B-C.
007	022	**STAX D**	Store A at D-E.
010	166	**HLT**	Stop.

Step 12. Execute the program at START (003000_8).

Step 13. Select address 003030_8.

Question. What is the data contained at this address? ____________________(333)
NOTE: If you want to verify this data again, change the contents of location 002000 to some other value.

Question. Which location or locations would you have to modify if you wanted to load A from 002010_8?

____

____

____

(The only location modified would be 003001_8, the low-order address byte for B-C. It would change from 000_8 to 010_8.)

Question. Which location or locations would you have to modify if you wanted to store A at location 002050_8?

____

____

____

____

(You would have to modify both location 003004_8 and 003005_8. Location 004 would become 050_8, and 005 would become 002_8.)

Question. Which location or locations would you have to modify if you wanted to store 030_8 instead of 333_8?

(You would have to modify location 0020000_8 to 030_8 instead of 333_8.)

Question. Why are the **LDAX rp** and **STAX rp** instructions classified as indirect addressing modes?

(Because the instruction specifies a register pair that contains the address in memory where the data is or where it will be located.)

Immediate Addressing

The instructions **MVI r, data**, **LXI rp, data 16**, **ADI data**, and **SUI data** are examples of the immediate addressing mode. With these instructions, the data immediately follows the op code. The **MVI r, data** must also specify the register (in the op code) where the following data is to be placed. Bits 3, 4, and 5 are used to specify the register. They are identified as DDD (for destination); later you will also find that you have to specify a register in which data is contained. These will be identified as SSS (for source). The bit patterns for either the destination or source registers are found in the "Symbols and Abbreviations" section in the appendix entitled "8080 Instruction Set." Look at the portion labeled DDD,SSS under that section.

The **LXI rp, data 16** must specify the register pair where the following data is to be located. Again the RP (register pair) is identified by bits 4 and 5 of the op code. You should already be familiar with these bit patterns.

In the previous program (step 11) you used the **LXI rp, data 16** instruction to load the address in a register pair for the **LDAX** and **STAX** instructions. You could have used the **MVI r, data** to accomplish the same thing. For example:

Step 14. Enter the following program starting at location 003000_8.

Location	Code	Mnemonic	Comments
003000	016 (START)	**MVI C**	Load C with data.
001	000		
002	006	**MVI B**	Load B with data.
003	002		
004	036	**MVI E**	Load E with data.
005	030		
006	026	**MVI D**	Load D with data.
007	003		
010	012	**LDAX A**	Load A from B-C.
011	022	**STAX A**	Store A at D-E.
012	166	**HLT**	Stop.

Step 15. Execute the program at START.

Question. What are the contents of location 003030?____________________(333)

As you can see, this program accomplished the same thing as the previous program. However, it required two extra locations and two extra memory accesses. Therefore, it took slightly longer to execute (because of the fast execution time, you will not notice the extra time required).

Step 16. Enter the following program starting at address 003000_8.

Location	Code	Mnemonic	Comments
003000	076 (START)	**MVI A**	Load A with data.
001	006		
002	306	**ADI**	Add to A data.
003	006		
004	326	**SUI**	Subtract from A data.
005	003		
006	323	**OUT**	Output A
007	000		to Port 0.
010	166	**HLT**	Stop.

Step 17. Execute the program at START.

Question. Describe the operation of this program.

____

____

____

____

____

(This program will move the data value 6 to the accumulator. Then it will add the value 6 to the contents of the accumulator. Then the program will subtract the value 3 from the accumulator, output the result 9_{10} (011) to output Port 0, and stop.)

Question. What is the content of output Port 0?__________________(011_8 or 9_{10})

Register Addressing

The **MOV r1, r2,** and **ADD r** are examples of register addressing techniques. In both of these instructions, the register in which the data is contained is called the source (S). The source register is identified by bits 0, 1, and 2 of the instruction. You should already be familiar with these bit patterns. The **INR r** and the **DCR r** are also examples of register addressing. The first instruction increments the register (adds 1); the second decrements it (subtracts 1).

The following program will load the B and E registers with data. Move this data from each register to the A register and output the contents of B to Port 0 and E to Port 1.

Step 18. Enter the following program, starting at address 003000_8.

Location	Code	Mnemonic	Comments
003000	006 (START)	**MVI B**	Load B with
001	303		data.
002	036	**MVI E**	Load E with
003	111		data.
004	170	**MOV A,B**	Move B to A.
005	323	**OUT**	Output
006	000		Port 0.
007	173	**MOV A,E**	Move E to A.
010	323	**OUT**	Output
011	001		Port 1.
012	166	**HLT**	HALT.

Step 19. Execute the program at START.

Question. What are the contents of Ports 0 and 1? ______________________________
______________________________ (Port 0 = 303, Port 1 = 111)

Branch or Jump Instructions.

As mentioned earlier, branch or jump instructions cause the program to continue from some other memory location.

These jump instructions are either *conditional* or *unconditional.* The conditional instructions require the state of the microprocessor flags to be examined. These flags are zero, sign, parity, and carry. There is an additional flag which is not examined and that is the auxiliary carry. Bits 3, 4, and 5 of the instruction determine which flag is to be examined. The coding of these bits is found in the appendix entitled "8080 Instruction Set" in the section "Branch Group."

The unconditional jump will perform the program jump to the new location when this instruction is encountered in the program. Jumps are either direct, in which the following 2 bytes of the program specify the new location, or indirect, in which a register pair will contain the new address. We will only look at the direct jumps at this time.

The following program will start by clearing the A register. Increment it by 1, output, and jump to location 003020. At this location, A will increment, output, and jump to location 003050. At this location, A will increment, output, and the program will stop.

Step 20. Load the following program at location 003000_8. Note that you have three starting locations - 003000, 003020, and 003050.

Location	Code	Mnemonic	Comments
003000	257	**XRA**	Clear A.
001	074	**INR A**	Increment A.
002	323	**OUT**	Output to
003	000		Port 0.
004	303	**JMP**	Jump to location
005	020		003020.
006	003		
003020	074	**INR A**	Increment A.
021	323	**OUT**	Output to
022	001		Port 1.
023	303	**JMP**	Jump to location
024	050		003050.
025	003		

Location	Code	Mnemonic	Comments
003050	074	**INR A**	Increment A.
051	323	**OUT**	Output to
052	002		Port 2.
053	166	**HLT**	Stop.

Step 21. Execute the program at 003000.

Question. What are the contents of Ports 0, 1, and 2?
Port 0________, Port 1________, and Port 2 ________________ (1, 2, 3)

Step 22. Modify the program starting at address 003053. Load the following instructions.

003053	303	**JMP**	Jump to location
054	020		003020.
055	003		

Step 23. Execute the program at address 003000.

Question. Describe the operation of this program.

____

____

____

____

____

____

____

____

____

(This program starts by clearing the A register, then increments it by 1, outputs to Port 0, then jumps to location 003020. At this location the A register is incremented by 1, outputs to Port 1, and jumps to location 003050. At this location the A register is incremented by 1, outputs to Port 2, and jumps to location 003020. The program is in a loop between locations 003020 and 003050.)

Step 24. Load the following program at location 003000.

Location	Code	Mnemonic	Comments
003000	076 (START)	**MVI A**	Move the following
001	377		data to A.
002	075	**DCR A**	Decrement A.
003	323	**OUT**	Output to
004	000		Port 0.
005	302	**JNZ**	Jump to location
006	002		003002 if not equal 0.
007	003		
010	315	**CALL**	Get the subroutine
011	030		at location 003030.
012	003		
013	303	**JMP**	Jump to location
014	000		003000.
015	003		

Step 25. Load the following program, starting at location 003030.

Time Delay Subroutine

Location	Code	Mnemonic	Comments
003030	365	**PUSH PSW**	Store the value of the stack pointer (SP) register pair.
003031	325	**PUSH D**	Store the value of the D-E register pair.
003032	021	**LXI D**	Enter the following data
003033	046		into E,
003034	222		into D.
003035	033	**DCX D**	Decrement D-E by 1.
003036	172	**MOV A,D**	Move register D to A.
003037	263	**ORA E**	OR registers E and A.
003040	302	**JNZ**	If not equal to 0, go back
003041	035		to location 003035.
003042	003		
003043	321	**POP D**	Return the D-E value memory.
003044	361	**POP PSW**	Return the SP value from memory.
003045	311	**RET**	Return to the main program.

Step 26. Execute the program at START.

Question. Describe the apparent action of output Port 0. ________________ (flashing)

Step 27. RESET and turn off the trainer.

Summary

In this activity you entered and executed the various addressing techniques of the 8080 microprocessor. These techniques or modes included direct, indirect, immediate, and register. You also entered direct jump or branch instructions, both conditional and unconditional types. These branch instructions are also considered an addressing mode

In this activity you learned there are various ways to retrieve or get information from memory. You also learned how to move this data around.

You learned that the direct method gives the address of the data, the indirect method references a register pair for the address, and the address must be pre-loaded into the pair.

You also learned that the immediate technique has the data following the instruction and the register technique references a register or register pair for the data.

You were also introduced to the branch instructions where direct jumps were concerned.

Registers Lab

Purpose

The purpose of this laboratory is to provide you with experience in moving data into and out of the various 8080 microprocessor registers.

Objectives

- To code simple programs involving register selection and data movement

Equipment

- MMD-2 Mini-Micro Design/Trainer

Introduction

Registers are temporary storage elements of any computer system. The capacity of these registers is usually equal to one computer word. The 8080 microprocessor contains six 8-bit general purpose registers, an 8-bit accumulator (known as the A register), a 16-bit stack pointer register, and a 16-bit program counter register.

The general purpose registers are A, B, C, D, E, H, and L. These registers can be used as a single 8-bit (1-byte) register or used in pairs to form three 16-bit registers. When used in pairs, the combination B-C, D-E, and H-L is the register pair (RP).

To select a specific register or register pair, a code for that register is required in the operation code (op code). The octal code assignments for the registers are:

Register	Code
B	0
C	1
D	2
E	3
H	4
L	5
A	7

The code assignments for specific register pairs are:

RP	Code
B-C	0
D-E	1
H-L	2
Stack Pointer (SP)	3

The location of the code bits within an op code are identified as DDD, SSS, or RP.

DDD stands for the destination register (where you want the data to be upon completion of the instruction). SSS stands for the source register (from where you expect to receive or get data). RP stands for register pair and can be either the destination or source, depending on the instruction itself.

Starting with this lab, you will have to code part of the given programs. You will be given the mnemonic for the instruction, and you will have to look up the code for that mnemonic in the 8080 instruction set. The instruction set is located in the appendix of this lab manual. Then you will enter, execute, and verify the programs using the MMD-2 trainer.

You will also use the REG (Register) and STP (Step) FUNCTION keys to look at the data contained in these registers.

Procedure

General Registers

Step 1. Complete the coding of the following program.

NOTE: Be sure you start with the following functions/modes set MEM (Memory) and PORTS OFF. (Compare your program with that listed in the Answer Key located at the end of this lab.)

Code	Mnemonic	Comments
____	**MVI B**	Move the following to B.
377		Data to B.
____	**MOV A,B**	Move contents of B to A.
____	**OUT**	Output
000		to Port 0.
____	**HLT**	Halt.

Step 2. Enter the program you just coded, starting at address 3000_8.

Step 3. Execute the program you just entered. (NOTE: If your program failed to operate properly, compare your codes with those given in the Answer Key located at the end of this lab.)

Question. Describe the operation of this program.

(Moves the data contained in the following location to the B register. The B register is then moved to the A register. The program then outputs the contents of the A register to Port 0 and stops.)

Question. What are the contents of output Port 0? (377)

Question. If you had not moved the data from B to A, would Port 0 equal the data?

(no)

Question. Why?

(Because the output instruction transfers (moves) the data contained in the A register to output port. Because the data was not moved from register B to A, the output data would not equal the desired data.)

Step 4. Select the REG (Registers) function, register A.

Question. What is the data contained in A? (377)

The following program will move data from one register to another until all general registers have been affected. Each time you go to a different register, it will be incremented with the last register (A) outputting the count. Therefore, the output count should equal the number of general registers in the 8080.

Step 5. Complete the coding of the following program. (Compare your program with that listed in the Answer Key located at the end of this lab.)

Location	Code	Mnemonic	Comments
003000	____(START)	**MVI B**	Move the following to B.
001	000		Data to B.
002	004	**INR B**	Increment B.
003	____	**MOV C,B**	Move contents of B to C.
004	014	**INR C**	Increment C.
005	____	**MOV D,C**	Move contents of C to D.
006	____	**INR D**	Increment D.
007	____	**MOV E,D**	Move contents of D to E.
010	____	**INR, E**	Increment E.
011	____	**MOV H,E**	Move contents of E to H.
012	____	**INR H**	Increment H.
013	____	**MOV L,H**	Move contents of H to L.
014	____	**INR L**	Increment L.
015	____	**MOV A,L**	Move contents of L to A.
016	____	**INR A**	Increment A.
017	____	**OUT**	Output
020	000		to Port 0.
021	____	**HLT**	Halt.

Step 6. Select MEM (Memory) function and enter the program at START.

Step 7. Use the REG (Registers) function key and observe the contents of each of the registers as you STP (Step) through this program. NOTE: Remember, the order for the register function is A,B,C,D,E,H,L,SH,SL,FL. The registers order for this program is B,C,D,E,H,L, and A.

Question. What are the contents of the A register after step 15?________________(6)
After step 16?_____________(7)

Question. What are the contents of Port 0 after step 20?______________________________
(007_8=7_{10})

Register Pairs

As stated earlier, the general registers can be combined into register pairs. When this combination is requested, 16 bits of information are available. The register pairs are identified as B (general registers B and C), D (general registers D and E), H (general registers H and L), and a 16-bit register called stack pointer (SP). The stack pointer register will be used to designate an area of the R/W memory (RAM) that will store information for certain instructions. These instructions are called **PUSH** and **POP**. The KEX MONITOR PROM has designated an area as the stack area for the MMD-2.

The following program will load data into the B register pair, then output the data.

Step 8. Select the MEM (Memory) function and load the following program, starting at 3000_8.

Location	Code	Mnemonic	Comment
003000	001 (START)	**LXI B**	Load RP immediate.
001	345		Low-order data.
002	012		High-order data.
003	171	**MOV C,A**	Move C to A.
004	323	**OUT**	Output
005	000		Port 0.
006	170	**MOV B,A**	Move B to A.
007	323	**OUT**	Output
010	001		Port 1.
011	166	**HLT**	Halt.

Step 9. Execute the program at START.

Question. What are the contents of Port 1? ________________ (012_8)

Question. What are the contents of Port 0? ________________ (345)

Question. Why was it necessary to have the two **MOV** instructions?

____

____

____

(Because the accumulator is 1 byte (8 bits) and in order to output the data in RP-B, each byte had to be moved to the accumulator separately.)

Question. Which general register (B or C) contains the low-order byte? ________ (C)

Question. Which general register (B or C) contains the high-order byte? ________ (B)

The following program will output the contents of the SP register. To do this, you will have to clear the H-L register pair, add the contents of SP and HL, then output H-L. The reason for using the H-L is that the SP register is not directly accessible to the A register.

Step 10. Load the following program, starting at 3000_8.

Location	Code	Mnemonic	Comments
003000	041 (START)	**LXI H**	Load H-L with
001	000		low-order data,
002	000		high-order data.
003	071	**DAD SP**	Add RP to H-L.
004	175	**MOV L,A**	Move L to A.
005	323	**OUT**	Output
006	000		Port 0.
007	174	**MOV A,H**	Move H to A.
010	323	**OUT**	Output
011	001		Port 1.
012	166	**HLT**	Halt.

Step 11. Execute the program at START.

Question. What are the contents of the SP register (Ports 1 and 0)?

____ (003 376)

(Remember, you can "look" at the contents of the Stack Pointer Register by selecting REG (Registers) function and SH, SL. Don't forget to return back to the MEM function.)

Step 12. Code and enter the following program. Enter at START address 3000_8.

Location	Code	Mnemonics	Comments
003000	___ (START)	**LXI D**	Load RP D-E with
001	000		low-order data.
002	000		high-order data.
003	___	**MVI A**	Load A with
004	005		data.
005	___	**INX D**	Increment PR D-E.
006	___	**DCR A**	Decrement A.
007	302	**JNZ**	A not equal, jump to
010	005		low-order address,
011	003		high-order address.
012	___	**MOV A,E**	Move contents E to A.
013	323	**OUT**	Output
014	000		Port 0.
015	___	**MOV A,D**	Move contents D to A.
016	323	**OUT**	Output
017	001		Port 1.
020	166	**HLT**	Halt.

Step 13. Execute the program at START.

Question. What are the contents of Ports 1 and 0?

____ (000 005)

Question. Describe the operation of this program.

(This program loaded register pair D-E with zero. Then it loaded the A register with the quantity 5. It then incremented the RP and decremented A. It checked to see if A was equal to zero; if not zero, it jumped back to the increment RP instruction. When A equaled zero, it then moved the contents of E to A and output to Port 0; then it moved the contents of D to A and output to Port 1 and stopped.)

Summary

In this lab you coded, entered, and executed various programs dealing with 8080 registers and register pairs. You selected the correct register as required and the code for that register, which you then combined with the instruction to complete the program.

In this lab you learned that the 8080 microprocessor has six 8-bit registers in which data can be stored. You learned how to select these registers with a specific code. You also learned that these registers could be combined to form 16-bit register pairs and that there is a register called the stack pointer which contains a memory address for special operations. You also learned how to code the register pairs. All of this required you to select the specified register, code the instruction, enter, and execute programs.

Answer Key

```
Step 1.  006              MVI B
         377              (DATA)
         170              MOV A,B
         323              OUT
         000              (PORT 0)
         166

Step 5.  006              MVI B
         000              (DATA)
         004              INR B
         110              MOV C,B
         014              INR C
         121              MOV D,C
         024              INR D
         132              MOV E,D
         034              INR E
         143              MOV H,E
         044              INR H
         154              MOV L,H
         054              INR L
         175              MOV A,L
         074              INR A
         323              OUT
         000              (PORT)
         166              HLT

Step 12. 003000   021     LXI D
         001      000
         002      000
         003      076     MVI A
         004      005
         005      023     INX D
         006      075     DCR A
         007      302     JNZ
         010      005
         011      003
         012      173     MOV A,E
         013      323     OUT
         014      000
         015      172     MOV A,D
         016      323     OUT
         017      001
         020      166     HLT
```

Mnemonic Instructions Lab

Purpose

The purpose of this lab activity is to give you an opportunity to complete and execute on the trainer several simple machine language programs which utilize data transfer, arithmetic, and logical group instructions.

Objectives

- To complete or debug a program listing to solve a problem statement
- To enter and execute the program on the microprocessor trainer

Equipment

- MMD-2 Mini-Micro Design/Trainer

Introduction

As you know, the purpose of the DATA TRANSFER group of instructions is to move (transfer) data around in the 8080 microprocessor. This data can be moved between memory and the microprocessor and also between registers within the microprocessor.

Likewise, the purpose of the LOGICAL group is to perform logical operations (AND, OR or X-OR) on the data. To be able to do this, you need two sets of data. One set you select as a standard and the other set would be "unknown."

One of the results of this type of operation is called masking, in which you can hide or mask unwanted data. For example, if the data byte input contained the actual data and several status bits, the status bits would be interpreted as data during calculations unless they were masked out. This would result in an incorrect data processing. Let's look at an example of masking. If the lower 4 bits of the data byte were for data and the upper 4 bits for status, you could input **IN** the byte and **ANI** (AND immediate) the following byte 11110000. Since a zero on an AND function always results in zero, the lower 4 bits would always be zero (masked). The upper 4 bits would be either 1 or 0, depending on the input data.

Some instructions from the data transfer and arithmetic groups can be interchanged with each other to produce the same results. For example, if you wanted to place all zeros in the A register you could either move immediately all zeros into A (**MVI A, data**), which requires two memory accesses, or you could EXCLUSIVE OR the A register with itself (**XRA A**). The **XRA A** requires one memory access. Remember the X-OR logic truth table states: like terms = 0, unlike terms = 1.

Finally, the ARITHMETIC group instructions allow you to perform arithmetic operations, such as add and subtract, on two sets of data. Remember, multiplication is only a form of successive adds, and division is successive subtracts.

Note that you will also use several BRANCH instructions during this activity.

Procedure

NOTE: The following steps require you to locate and code the various instructions. The listing for these instructions is found in the appendix of this manual. Note also that no specific step will tell you to use the REG (Register) function. However, feel free to use this function (in fact, it is encouraged) to observe the contents of the various registers you select to use. Don't forget to return to the MEM (Memory) function before attempting to alter, modify, or change your program instructions.

Data Transfer

Step 1. Write a routine to load (move) the number 10_{10} into the B register; then output the contents of B to Port 1 and stop. Your program should consist of six instructions.

Address	Code	Mnemonic	Comments
003000			
003001			
003002			
003003			
003004			
003005			

Step 2. Enter and execute your program.

The easiest way to accomplish this program is to move immediate to B (**MVI B, data**) the data (don't forget to convert the data from decimal to octal). Since you cannot output data from the B register, you will have to move it from B to A (**MOV A,B**), output (**OUT**) and stop (**HLT**) (codes = 006, 012, 170, 323, 001, 166).

Step 3. Write a program that will load the accumulator directly from a memory location, output this data to Port 2 and halt. The address containing the data will be 002000_8. This program will require six instructions plus data location.

Address	Code	Mnemonic	Comment
002000	012	(data)	Data to be loaded.
003000			
003001			
003002			
003003			
003004			
003005			

Step 4. Load and execute the program.

The easiest way to accomplish this program is to load the accumulator directly from the memory address as specified (**LDA addr**). Since the data is contained in the register already, you can output at this time (**OUT**) and stop (**HLT**) (codes = 072, 000, 002, 323, 002, 166). You will have to load location 002000_8 with the data (12_8) prior to execution of the program.

Step 5. Write a program that will load register pair H,L with data; then output the contents of L to Port 1 and the contents of H to Port 2; then stop. The data to be loaded is 12_8 in both registers. This program will require ten locations.

Address	Code	Mnemonic	Comments
003000			
003001			
003002			
003003			
003004			
003005			
003006			
003007			
003010			
003011			

Step 6. Enter and execute the program.

The easiest way to accomplish this program is to load the register pair H,L with the immediately following data (**LXI rp, data 16**). Again, since you cannot output directly to a port from these registers, the program will have to move each byte of data to A and then output. (**MOV r_1, r_2**) (codes = 041, 012, 012, 175, 323, 001, 174, 323, 002, 166).

Arithmetic

Step 7. This program adds together two numbers and outputs the results.

Address	Code	Mnemonic	Comments
3000	076	**MVI A**	Move 5 to A.
3001	005	005	
3002	___	**MVI B**	Move 1 to B.
3003	001	001	
3004	___	**ADD B**	Add.
3005	323	**OUT**	Output sum.
3006	001	001	
3007	166	**HLT**	Stop.

Question. Fill in the code required at addresses 3002 and 3004.

______________________________ (006, 200)

Step 8. Enter and execute the program.

Question. What result is displayed at Port 1? ______________________________

Is this what you expected? ______________________________ (00000110, yes)

Step 9. Modify the program to add 5 + 6. Execute it.

Question. What modifications did you make in the program to add 5 + 6? ______________

______________________________ (modify location 3003 to read 006)

Question. What result is now displayed in Port 1?

______________________________ (00001011)

Step 10. Modify the program to subtract 5 minus 1 and execute it.

Question. List the subtraction program you have just written and the result displayed in Port 1.

Address	Code	Mnemonic	Comments

(Program is the same as add except locations 3003 = 001 and 3004 = 220, result = 00000100.)

Step 11. Now modify the program to subtract 5 minus 6 and execute it.

Question. What are the results? Are the results correct? If not, what is wrong? ________

____

____

____

____

(Result: 1111 1111. This does not seem to be the binary code for -1. The problem is that the subtraction resulted in a borrow which transferred through all digits making them all ones.)

The result of this program indicates a situation that one has to be aware of during programming. During arithmetic operations, such as subtraction, if the result is negative, the microprocessor automatically changes the representation of the number from regular binary notation into 2's complement notation. Thus, in the last problem, one would have to know whether the result in Port 1 represented a number in regular binary code (in which case, the result is 255_{10}) or in 2's complement notation (in which case the result is -1).

The way to determine whether the result of a subtraction resulted in a negative number is to check the *Sign Flag* in the status register. The sign flag will be set if the result of an operation is negative. Various instructions such as jump allow you to branch to a specified memory location if the sign flag is at logic 1 (set). This location could be a routine that takes into account the changed representation of a negative number.

Let's try to do this in the following way. If the result of a subtraction is negative, let's convert the result back to regular binary notation and send this to Port 1. To indicate it is negative, let's output a 1 to Port 0. Thus -1 would be represented by bit 0 of both Port 1 and Port 0 being set. For any positive result, the answer is seen at Port 1, with no bits being set at Port 0.

A routine to handle the negative numbers will consist of two parts:

a. Convert from 2's complement notation to regular binary notation. This is done by complementing the number and adding 1 to it.

b. Set bit 0 of Port 0 by outputting a 1 there.

Step 12. Write a routine that accomplishes both of the above steps. Begin the routine at address 3020. (Compare your program with that listed in the Answer Key located at the end of this lab.)

Address	Code	Mnemonic	Comments

Step 13. Modify the main program to contain an instruction that tests the sign bit and branches to address 3020 if the bit is set. List the modified main program. (Compare your program with that listed in the Answer Key located at the end of this lab.)

Address	Code	Mnemonic	Comments

Step 14. Enter the entire program and execute it. The result should be a 1 in both Port 0 Port 1. If this is not the case, check over your work.

Step 15. Now try these problems by making the proper changes to your program.

6 - 8
3 - 10
9 - 4

Logical

The Logical Group instructions are those which program Boolean-type operations on data in the registers, memory, and condition flags. One of the advantages of these instructions is that they allow you to output or perform other operations on specific data bits. For example, you can *compare* 1 byte of data with another, or you can AND 2 bytes of data and output only those bits that are identical.

Step 16. Enter and execute the following program.

Address	Code	Mnemonic	Comment
003000	076	**MVI A**	Move 111 to A.
003001	111		
003002	006	**MVI B**	Move 111 to B.
003003	111		
003004	240	**ANA B**	AND B with A.
003005	323	**OUT**	Output to Port 0.
003006	000		
003007	166	**HLT**	Stop.

Question. What are the results displayed in Port 0? ______________________ (111)

Step 17. Modify the program by changing the contents of location 003003 to read 222.

Question. What are the results displayed in Port 0 now? ______________________ (000)
Question. Explain the results of Step 17.

____

____

(The program was an AND program, which was to AND the contents of A [01001001] with the contents of B [10010010]. Since 1 AND 0 logically equal 0, all zeros are displayed in Port 0.)

Step 18. Modify the program by changing the **ANA B** instruction to an **ORA B** instruction (code = 260) and execute.

Question. What are the results displayed in Port 0 now? ______________________ (333)

Question. Explain the results of step 18. ______________________________________

____

(OR function, therefore any 1 in will produce a 1 out)

Another function of the logical instructions is called *masking.* Masking is the concept whereby certain bits in the data word are cancelled (masked). For example, the data byte from an external device might contain status (such as POWER-ON and READY) as well as data. You would not want to process the status portion, because it would result in incorrect results. Therefore, you mask these bits. To mask the bit positions is fairly simple. In the bit positions we wish to mask out, place zeros; in the bit positions we wish to keep, place ones.

For example, the input word we receive is 370_8, with the upper 4 bits being status. To mask these positions, the masking word would be 017_8.

Step 19. Enter and execute the following program, which will mask the upper 4 bits from an input data word and add the quantity five.

Address	Code	Mnemonic	Comment
003000	076	**MVI A**	Move 371 to A.
003001	371		
003002	346	**ANI**	AND 017 to A.
003003	017		
003004	306	**ADI**	Add 5 to A.
003005	005		
003006	323	**OUT**	Output to Port 0.
003007	000		
003010	166	**HLT**	Stop.

Question. What are the results as displayed in Port 0? ______________________ (016)

Step 20. Change the contents of locations 003002 and 003003 to 000 **(NOP)**.

Question. What are the results as displayed in Port 0 now? __________________ (376)

The **NOP** instruction allows you to continue processing without having to reprogram. You have seen the results of both programs operating on the same data. The first, however, masked out the upper 4 bits; the second did not.

In this program we will attempt to produce a "marching light" effect on the LED outputs on Port 1. We would like to light the LEDs in order from right to left along all 8 bits.

To do this, we will put a 1 in the A register and output it to Port 1. Then we will shift this bit one place left and output the result to Port 1. We will execute this operation in a continuous loop.

Address	Code	Mnemonic	Comments
3000	076	**MVI A**	Put 1 in A register.
3001	____	______	________________
3002	____	______	Output to Port 1.
3003	____	______	________________
3004	____	**RLC**	Rotate A reg bit left.
3005	____	**OUT**	Output to Port 1.
3006	____	001	________________

All mnemonics copyright Intel Corporation 1977 and 1975

3007	____	**JMP**	Jump back
3010	____	______	and rotate again.
3011			

Step 21. Complete the program shown above. Enter and execute the program.

Question. What happens to the LEDs at Port 1?

____ (all lit)

To slow things down a bit, we need to add a time delay before outputting the data each time. This is easily accomplished in a subroutine. We will use a subroutine similar to the one illustrated in the lab that introduced you to the trainer. (The contents of A register are preserved by this routine so that we don't change what we want to output each time.)

10-MS TIME DELAY

Address	Code	Mnemonic	Comments
3020	107	**MOV B,A**	Save A register.
3021	021	**LXI D**	Load into D-E reg pair.
3022	046	046	046
3023	001	001	001
3024	033	**DCX E**	Decrement D-E by 1.
3025	172	**MOV A,D**	Move D to A.
3026	263	**ORA**	OR register E and A.
3027	302	**JNZ**	Jump not zero.
3030	024	024	
3031	003	003	
3032	170	**MOV A,B**	Restore A register.
3033	311	**RET**	Return.

As you remember, the contents of addresses 3022 and 3023 control the length of the time delay.

To enter the delay subroutine, all you need do is call 3020 from the main program.

Step 22. Add the proper **CALL** statement to the main program. List the main program with the **CALL** statement included. (Compare your program with that listed in the Answer Key located at the end of this lab.)

Address	Code	Mnemonic	Comments

Step 23. Execute the program.

Now, at least, the LEDs can be seen flashing. (If they don't, check to see that the program was entered correctly.)

Question. To slow the display flashing rate even more, the contents of which addresses must be changed?

____ (3022, 3023)

Step 24. Modify the contents of those addresses until you obtain a display speed you like.

Step 25. Modify the program so that the display starts at the left and moves to the right (only the contents of two addresses need changing).

Question. List the addresses and contents that you modified.

____ (3001 200 data)

____(3004 017 **RRC**)

In this lab, you completed and/or debugged, entered, and executed different programs involving arithmetic, data transfer, and logic instructions.

In this lab, you learned about masking techniques and the representation of negative numbers inside the microprocessor. You also learned how to test the condition of a status flag and branch accordingly, and how to control the rate of execution of a program to allow the output of the program to be easily observed.

ANSWER KEY

Step 12.

```
3020          057    CMA      Complement A register.
3021          306    ADI      Add one to it.
3022          001    001
3023          323    OUT      Output to Port 1.
3024          001    01
3025          076    MVI A    Put 1 in A register.
3026          001    001
3027          323    OUT      Output to Port 0.
3030          000    0
3031          166    HLT      Stop.
```

Step 13.

```
3000          076    MVI A    Move 5 to A.
3001          005    005
3002          006    MVI B    Move 6 to B.
3003          006    006
3004          220    SUB B    Subtract.
3005          372    JM       Jump if minus sign.
3006          020    020      If not,
3007          003    003      then
3010          323    OUT      out.
3011          001    001
3011          166    HLT      Stop.
```

Step 22.

```
3000          076    MVI A
3001          001    001
3002          323    OUT
3003          001    001
3004          007    RLC
3005          315    CALL
3006          020    020
3007          003    003
3010          323    OUT
3011          001    001
3012          303    JMP
3013          004    004
3014          003    003
```

Instruction Data Flow Lab

Purpose

This activity covers 1-byte, 2-byte, and 3-byte instructions. You will be provided with a given program from which you will determine if the given code is a 1-byte, 2-byte, or 3-byte instruction or data. Then you will verify the given program by executing it on an MMD-2 trainer. Next, you will modify the program so that it produces errors. Finally, you will be given a problem statement and partial program listing to complete the program, and will enter, execute, and verify its operation on an MMD-2 trainer.

Objectives

- To identify 1-byte, 2-byte, and 3-byte instructions and their format
- To identify errors produced by improper program coding
- To complete a given program
- To enter, execute, and verify a program

Equipment

- MMD-2 Mini-Micro Design/Trainer

Introduction

As you work more closely with computer systems, you will become more aware that computers cannot think. Although computers can perform very complex operations within seconds, they must be told when and how each step is to be done. This "handholding" is done by the computer program, the sequential listing of instructions that directs the computer to perform a specific process. The instructions will tell the computer where or what the information is, what to do with the information, and what to do with the end result.

"Fine," you say, "but what does this have to do with me? I am going to be working with hardware, not software." The answer to that question is basically this: Whether you are working with a design team on a new system or are attempting to troubleshoot a system you must be able to answer the following questions about the system:

- What is its overall purpose?
- What is it not doing correctly?
- Is it receiving proper information?

- Is it processing information?
- Is it transmitting proper information?
- Where should it obtain information?
- Where should it place information?

To answer these questions, you must be familiar with the computer's instruction data flow, which will tell you what the system should do, where or what the information is, and where to obtain or place the information.

Procedure

You will need to refer to the 8080 Instruction Set in the appendix of this manual as you proceed through this lab.

Step 1. Complete the following program listing by filling in the Mnemonic and Comments columns: (See figure 1 for a flowchart of step 1. Compare your answers with the Answer Key located at the end of this lab for this step.)

Location	Code	Mnemonic	Comments
003000	076	________	________________
3001	000	________	________________
3002	006	________	________________
3003	100	________	________________
3004	016	________	________________
3005	100	________	________________
3006	015	________	________________
3007	302	________	________________

Location	Code	Mnemonic	Comments
3010	006	________	____________________
3011	003	________	____________________
3012	005	________	____________________
3013	302	________	____________________
3014	004	________	____________________
3015	003	________	____________________
3016	323	________	____________________
3017	001	________	____________________
3020	323	________	____________________
3021	002	________	____________________
3022	075	________	____________________
3023	303	________	____________________
3024	002	________	____________________
3025	003	________	____________________

Question. Which locations contain 1-byte instructions?

______________________________ (3006, 3012, 3022)

Question. Which locations contain 2-byte instructions?

______________________________ (3000, 3002, 3004, 3016, 3020)

Question. Which locations contain 3-byte instructions?

______________________________ (3007, 3013, 3023)

Question. Which 2-byte instructions contain data in the second byte?

______________________________ (000, 002, 004)
(Remember, in location 3021 is the address of the output port, not data.)

Question. Which 2-byte instructions contain address data in the second byte?

____ (3016, 3020)

Question. Which 3-byte instructions contain raw data in the second and third bytes?

____ (none)

Question. Which 3-byte instructions contain address data in the second and third bytes?

____ (3007, 3013, 3023)

Question. Describe in detail the operation of this program.

____

____

____

____

____

____

____

____

____

____

____

____

(This program starts by moving zeros to the A register, 100_8 to the B register, and 100_8 to the C register. Then it decreases the C register by 1 until the C register equals zero [loops]. When the program detects zero in the C register, it continues by decrementing the B register by 1. If the B register does not equal zero, the program jumps back and loads the C register with 100_8 and repeats the C register loop. This "loop within a loop" continues until the B register equals zero. At that time the program outputs the contents of the A register to Port 1, then Port 2. The program then decrements the A register by 1 and jumps back to location 003002 and repeats the process.)

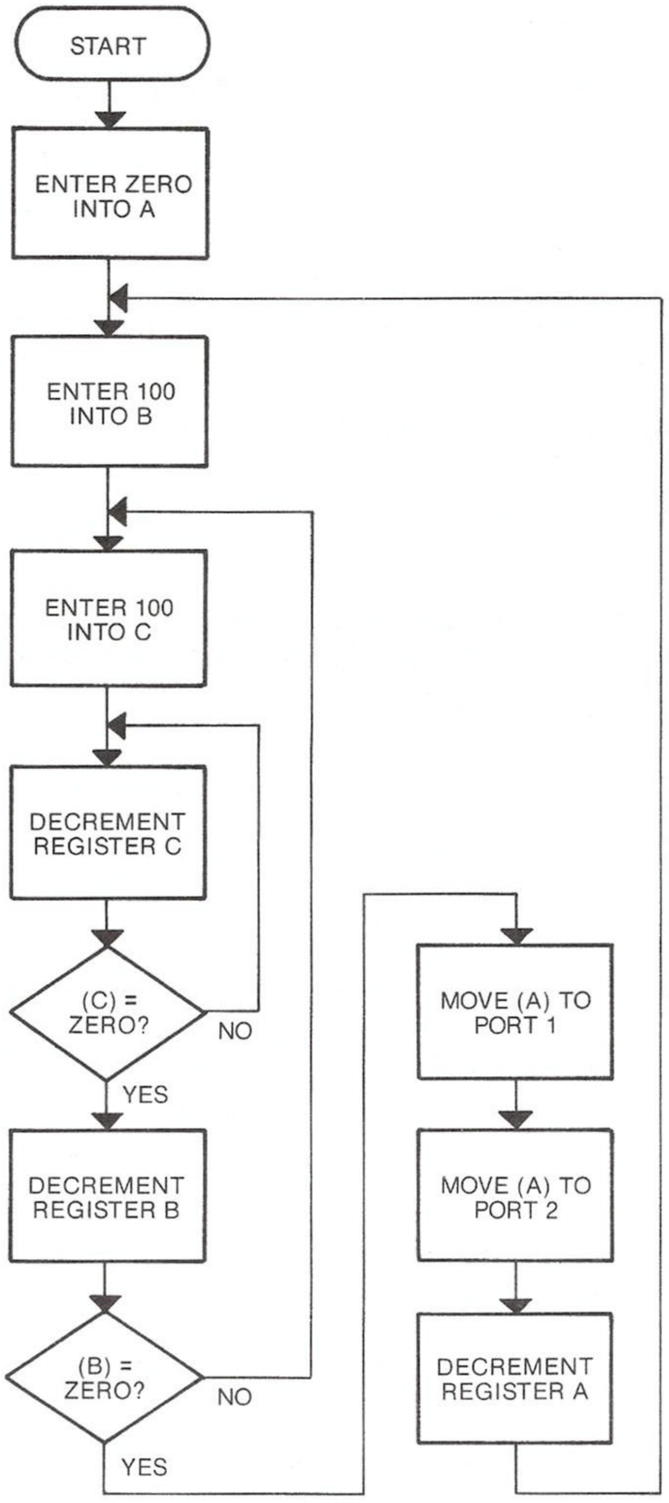


Figure 1. Flowchart for Program to Decrement Output

Step 2. Enter and execute the program using the MMD-2.

Question. Are the indicators in Ports 1 and 2 decrementing?

______________________________ (if not, verify program)

Step 3. Modify the program by changing the contents of location 3003 to read 000 and execute the program at 003000.

Question. The output indicators are decrementing at a ______________________ (slower/faster) rate. (slower)

Step 4. Modify the program by changing the contents of location 3005 to read 000 and execute the program at 003000.

Question. The output indicators are decrementing at a ______________________ (slower/faster) rate. (slower)

Question. In this program, you have now moved zeros into both the B and C registers. Locations 3007 and 3013 will cause the program to jump back (loop) if B and C are not equal to zero and to output if they are equal. The indicators should appear to be constantly on. Why aren't they?

____

____

____

____

(Because both registers are decremented before being tested for zero. This decrement causes the registers to be 377 [000-1=377].)

Question. This program could be used as a ______________________.
(time delay)

You will now introduce some "errors" into the program and observe their effects. These errors could be the result of incorrect program entry (software) or hardware problems.

Step 5. Return the program to its original condition. Change the contents of locations 3003 and 3005 to read 100.

Step 6. Verify the program by executing it at 003000.

Step 7. Modify program contents of location 3002 to read 004. Execute the program at 003000. The purpose of this change is to "accidentally" change the instructions from 00 000 110_2 to 00 000 100_2 so that instead of loading B with 100_8 the program is incrementing B by one. You will find out what a difference a bit makes!

Question. Describe indicator performance now. ______________________________

____

(Upper indicators are changing very fast. It is not clear if they are incrementing or decrementing.)

Question. What instruction results from the "dropping" of bit 1? ______________
(increment B)

Step 8. Modify program contents of location 3002 to read 014. Execute the program at 003000.

Question. Describe indicator performance now. ______________________________
(decrementing slowly)

(The new instruction 014_8 = 00 001 100_2 is INRC, a one-byte instruction. The old instruction was MVIB, a two-byte instruction. As you did not delete the second byte, located at 3003 [01 000 000_2], the PC thought that old data was the instruction MOVB, B a do-nothing instruction.)

Step 9. Modify program contents of location 3002 to read 006 and location 3010 to read 014. Execute the program at 003000.

Question. Describe indicator performance now. ______________________________

____ (constant 377 both ports)

Question. Describe the effect of this "bit shift" (programming error).

____

____

____

(This error causes the program to jump to location 3014 instead of looping back. At location 3014, the low-order address byte now becomes an instruction to increment the B register and, at location 15, to increment the register pair BC. The program will continue in this large loop.)

Step 10. Modify program contents of location 3010 to read 006, location 3024 to read 003, and location 3025 to read 002. Execute the program at 003000.

Question. Describe the performance of the program now.

____

____

____

____

____

____

(The program is jumping to location 002003, where the information is unknown. The result is that the program will not perform as expected and the LEDs will have various actions. This is an example of the result of misprogramming of the second and third bytes of a destination-type instruction.)

The preceding steps have shown you the problems that can arise from program entry error: "drop" bits, "add" bits, shift of bit position, and so forth. These problems can occur in either software or hardware.

For this final section, you are to modify the program you have been working with in this lab to meet the following parameters:

1. Clear A register with the EXCLUSIVE OR function
2. Load register pair DE with the same data as before; i.e. 100 and 100
3. Decrement the two registers (the register pair)
4. Output to Ports 1 and 2
5. Increment the A register

Step 11. Rewrite the program to meet the five parameters listed above. Enter and execute the program. NOTE: Your program should perform like the one listed. (Compare your program with the Answer Key located at the end of this lab for this step.)

Location	Code	Mnemonic	Comments (optional)
003000	______	______	____________
3001	______	______	____________
3002	______	______	____________
3003	______	______	____________

Location	Code	Mnemonic	Comments (optional)
3004			
3005			
3006			
3007			
3010			
3011			
3012			
3013			
3014			
3015			
3016			
3017			
3020			
3021			
3022			
3023			

Summary

In this lab, you identified the various 1-byte, 2-byte, and 3-byte instructions contained in a given program and the bytes containing raw data and address data. You then modified several locations to produce "errors." Finally, you rewrote the program to do the same job; however, it required fewer locations.

In this lab, you learned the format of 1-byte, 2-byte, and 3-byte instructions. You also learned whether the codes contained raw data or address data, which helped you determine the data flow of the program. You also learned the effect of hardware or software problems on a program's operation. Finally, by "modifying" a program, you have started programming with machine language instructions.

Answer Key

Step 1.

MVI A	Move (data) to A.
(data)	Data for A.
MVI B	Move (data) to B.
(data)	Data for B.
MVI C	Move (data) to C.
(data)	Data for C.
DCR C	Decrement C.
JNZ	If C not equal 0, go to
(data)	low address,
(data)	high address.
DCR B	Decrement B.
JNZ	If B not equal 0 go to
(data)	low address,
(data)	high address.
OUT	Output to
(data)	device number.
OUT	Output to
(data)	device number.
DCR A	Decrement A.
JMP	Jump to
(data)	low address,
(data)	high address.

Step 11.

257	**XRA A**
021	**LXI D**
100	
100	
035	**DCR E**
302	**JNZ**
004	
003	
025	**DCR D**
302	**JNZ**
001	
003	
323	**OUT**
002	
323	**OUT**
001	
074	**INR A**
303	**JMP**
001	
003	

Arithmetic Problem Lab

Purpose

The purpose of this lab activity is to provide you with some practice in completing or debugging some simple arithmetic programs and then executing them on the microprocessor trainer.

Objectives

After completion of this activity you will be able:

- To complete or debug a simple arithmetic program
- To enter and execute the program on a microprocessor trainer

Equipment

- MMD-2 Mini-Micro Design/Trainer

Introduction

During this lab activity you will either complete and/or debug different arithmetic programs. These programs will use the **ADD** or **DAD** instructions available in the 8080 microprocessor instruction set. Then each program will be entered on the MMD-2 trainer and executed to verify the results.

Procedure

Program 1

Write a program to add the decimal integers 3+7+12+18 and display the results.

This program will execute by fetching the numbers to be added one at a time from memory and accumulate the results in the A register. In order to know how many numbers there are to fetch, we will keep count of the numbers as they are fetched using the B register.

STORED DATA

2000	003	DATA, 3_{10}.
2001	007	DATA 7_{10}.
2002	___	DATA 12_{10}.
2003	___	DATA 18_{10}.

MAIN PROGRAM

3000	257	**XRA A**	Clear A registeı
3001	006	______	Put 4 into B
3002	004	004	register.
3003	041	**LXI H**	Load memory
3004	000	000	pointer into L and H.
3005	002	002	
3006	206	______	Add memory to A register.
3007	___	**DCR B**	Decrement count in B reg.
3010	___	**INX H**	Increment memory pointer.
3011	302	**JNZ**	Jump if count
3012	006	006	is not
3013	003	003	zero.
3014	___	**OUT**	Output results to
3015	001	001	Port 1.
3016	166	**HLT**	Stop.

Step 1. Complete the program by filling in the blanks. (Compare your answers with the Answer Key located at the end of this lab.)

A few comments about the program:

1) Notice that the memory location for the data is well away from the main program. This allows more data to be added later without having to juggle around other information. The program itself starts at location 3000_8. Since the data is not in the main body of the program, there is no danger of data being executed as an instruction; this can happen if data is included within the body of the program.

2) The number 4 is stored in the B register as a count of how many numbers to add.

3) The instruction at 3003 loads the memory location of the first piece of data into register pair HL. This register pair is used to "point" at the memory location that contains the data to be added to the A register. The **ADD M** instruction at 3006 always uses the memory location pointed at by the contents of HL.

4) The **DCR B** at 3007 keeps track of how many numbers have been added. Each time a number is added, the B register is decremented by 1. When the last number is added, the **DCR B** instruction will have caused the B register contents to be zero. Also at this time the Zero Flag will be set to 1; previously the Zero Flag had been reset to 0.

5) The **INX H** at 3010 changes the memory pointer in HL by 1 to point to the next memory location containing data.

6) The **JNZ** at 3011 checks the Zero Flag. As long as the Zero Flag is reset (logic 0) the program will go back to location 3006_8 to continue execution. When the Zero Flag is set, as it will be once the B register contents are zero, the **JNZ** instruction will be ignored.

Question. What binary word do you expect to see displayed at Port 1 after program has executed?

____ (00101000)

Step 2. Enter and execute the program. If the results are not as you predicted, check your work.

Step 3. Modify the program to perform the following addition:

3+7+12+18+21+15

Question. List the changes and additions you made.

____

____

____________________________________(2004 025 DATA,21)
(2005 017 DATA,15)
(3002 006)

Question. Predict the results to be displayed at Port 1.

____ (01001100)

Step 4. Enter the modifications into the trainer and execute. Check your work if the result is incorrect.

As you can see, this program is quite flexible for adding a list of numbers. Storing the data in sequential memory locations and using the **INX H** instruction make it easy for the program to index its way through the data to calculate the results.

You should be aware of certain limitations of this program. There is a maximum to the number of pieces of data which can be added by this program.

Question. What's the maximum number of pieces of data that this program can add together the way it is currently written?____________________

(256 since that is the number of memory locations between 2000_8 and 3000_8)

The next limitation has to do with the magnitude of the total.

Step 5. Modify the program to add the following list of numbers:

3+7+12+18+21+15+190

Step 6. Enter and execute the program.

Question. Are the results what you expected?____________________(no)

The problem here is not so much with a "bug" in the program but, rather, with a limitation of the program.

Question. Why are the results incorrect?

(The expected result is 266_{10} which is larger than the number that can be displayed. The highest number is $255_{10} = 11111111_2$.)

As you can see, even though we have a program that works properly on our initial data list, you must be careful when trying to apply it to all situations.

Along the same line as this second limitation, limiting the maximum total result, there is a third limitation that deals with the largest number that can appear in the data list.

Question. What is this number (base 10)?__________ (255)

A "Bug." An interesting "bug" can occur in this program if you reverse the instructions at locations 3006 and 3007.

Step 7. Change the program so that it will add the original list of numbers. Then reverse the code for the instructions at locations 3006 and 3007. Execute the program.

As you can see, the result is 0, which is obviously not correct. If you consider the logic of the program with the instructions reversed, at first sight there doesn't appear to be anything wrong. You must dig a bit deeper and examine exactly what each of these instructions does as it is executed.

Both the **DCR B** and the **ADD M** instructions affect the Zero Flag. The Zero Flag is important because, as long as it remains reset, the program will loop around, fetching data and adding it to the A register. When the instructions are reversed, even though the Zero Flag may become set by the **DCR B** instruction as the last piece of data has been fetched, the **ADD M** instruction will reset the flag as it is executed; hence, the program will continue execution even though all of the data has been added.

Yet, why do you suppose we get a result of 0 displayed at Port 1 every time this program is executed and not some random result? (Try it if you are skeptical.)

The answer is this: The program will continue adding random numbers obtained from memory locations past the end of our data until the result in the A register is 0. Once this occurs, the **ADD M** instruction also causes the Zero Flag to be set, and so the program will output the 0 to Port 1.

Therefore, you must be aware of all of the actions of an instruction as it is executed.

Program 2

Write a program that will add numbers together and allow the result to be larger than 255. Allow for up to 16 bits in your answer. Display the results in two output ports.

In this program we will use a special arithmetic instruction called the "double add" instruction, **DAD**. This instruction allows for 16-bit arithmetic.

3000	046	**MVI H**	Clear H.
3001	000	000	
3002	___	**MVI L**	Clear L.
3003	000	000	
3004	026	______	Clear D.
3005	000	000	
3006	006	______	Clear B.
3007	000	000	
3010	___	**LXI D**	Load memory location of
3011	000	000	first data item
3012	002	002	into DE.
3013	032	______	Load accumulator with data from address given in register pair DE.
3014	117	______	Move data to register C from accumulator.
3015	___	**ADI**	Check for flag
3016	001	001	by adding 1,
3017	332	**JC**	and jump
3020	026	026	if a
3021	003	003	carry occurs.
3022	011	**DAD B**	If no carry, add result to HL.
3023	023	**INX D**	Increment memory pointer.
3024	303	**JMP**	Jump back
3025	013	013	to this

3026	003	003	address.
3027	___	**MOV A,L**	Send L reg to A
3030	323	**OUT**	for output
3031	000	000	as LSB.
3032	174	________	Send H register to A
3033	323	**OUT**	for output
3034	001	001	as MSB.
3035	166	**HLT**	Stop.

Step 8. Complete the blanks in the program.

A few notes about this program:

1) Since the **DAD** instruction accumulates the results in register pair HL, these registers cannot be used as the memory pointer. Therefore, we will use register pair **DE** as the pointer.
2) The **DAD** instruction requires that the numbers to be added appear in another register pair. We have selected pair **DE**.
3) The **LDAX D** at 3013 puts the contents of the memory location pointed at by pair DE into the accumulator. Since this data is only 8 bits, it is stored by the **MOV C,A** instruction into register C.
4) Since all registers are being used by this program, we must use a method other than counting to know how many numbers to add together. This time we will use a flag. A flag is a piece of data appearing at the end of the list which signals that this is the end of the data. Our flag will be $377_8 = 11111111_2$. By adding one to the flag and checking for a carry (**JC** instruction at 3017), we can determine whether all of the data has been exhausted. Thus, the instructions from locations 3016 to 3020 check for the flag.
5) At 3022, the contents of register pair BC (which contain the data loaded from memory) are added, 16 bits at a time, with register pair HL. At 3023, the memory pointer is incremented by one and we jump back to load the next piece of data.
6) Once all of the data has been added, we branch to the output section of the program beginning at 3027 where the LSB of the result are output to Port 0 and the MSB output to Port 1.

To try this program out, we will add together two numbers that require more than 8 bits to display the result.

Let's add $254_{10} + 254_{10}$.

Question. Where in memory will you load each number, and what will you store there?

____(at $2000=376_8$)
(at $2001=376_8$)

Question. What will you store as a flag in the next memory location? (Remember the flag signals the end of the data.)____________________ (at 2002=377_8)

Step 9. Load the data and program into the microprocessor trainer memory.

Question. What result do you expect? Show your answer below.

Port 1								Port 0							
D7	D6	D5	D4	D3	D2	D1	D0	D7	D6	D5	D4	D3	D2	D1	D0

(00000001 11111100)

Step 10. Execute the program and verify the results.

Question. What modification must be made to add together:

254 + 254 + 254

____ (at 2002=376_8)

____ (at 2003=377_8)

Step 11. Make the modifications and execute the program.

Question. What is your result?
______________________________________ (00000010 11111010)

If your result is incorrect, check the data in memory.

Question. What is the largest number that can appear in the data list? Why?

____

____

____

____

____

(376_8 = 11111110_2 is the largest, because 377_8 is used as a flag and will not be treated as data. Numbers larger than this are not allowed, because the program only loads 8-bit words from memory into the BC register [see instructions at locations 3013 and 3014].)

As you can see, this program accomplishes much the same result as Program 1 but allows for a larger range of results and uses a different method to check for the end of the data.

Summary

In this lab you completed and/or debugged a variety of programs dealing with arithmetic problems. Once completed you entered the program and verified its result.

In this lab you learned about the arithmetic instructions **ADD**, **DAD**. You learned about the importance of the order of instructions when the status of the flags is important. You learned how to extend the range of the arithmetic possible with the 8080 by using the 16-bit double add instruction **DAD**. You learned how to use the instructions involving memory pointers and finally you learned how to keep track of items in a data list by either counting them or using a flag.

Answer Key

Step 1.

2002	014	
2003	022	
3001		**MVI B**
3006		**ADD M**
3007	005	
3010	043	
3014	323	

Step 8.

3002	056	
3004		**MVI D**
3006		**MVI B**
3010	021	
3013		**LDAX D**
3014		**MOV C,A**
3015	306	
3027	175	
3032		**MOV A,H**

Programming Lab

Purpose

In this activity you will generate and code a program to solve a problem.

Objectives

- To generate and code a program to solve a problem
- To enter and execute the program on a microprocessor trainer

Equipment

- MMD-2 Mini-Micro Design/Trainer

Introduction

This lab activity presents programming problems for you to solve. It is your goal to generate and code a well-documented program that will solve the problem. How you solve the problem will be up to you. However, if you model the way you go about it along the lines of good programming style (which you learned about earlier), you are likely to make fewer mistakes and to end up with an efficiently running program.

It is possible for you to take off on your own as soon as you read the problem statement and write your program. If you decide not to do this or get stuck along the way, a guide appears in the lab that suggests one way of proceeding. Feel free to use all or just parts of the guide as you proceed.

Procedure

Problem 1.

Write a program that uses the output ports to simulate a traffic light control. The time delays are "green" on for 30 seconds, "yellow" on for 10 seconds, and "red" on for 20 seconds. The operation is continuous, and no two colors are on at the same time.

(Space is provided for your work. The guide, if you need it, follows the blank programming sheet.)

Your General Outline

Your Refined Outline

Your Flowchart

Your Flowchart (Continued)

Your Assembly Language Program

Step Number	Label	Operand	Argument	Comments

Your Assembly Language Program (Continued)

Step Number	Label	Operand	Argument	Comments

Your Address Assignment

Your Address Assignment (Continued)

Your Symbol Table

Your Machine Language Program

Address	Code	Label	Mnemonic	Comments

Your Machine Language Program (Continued)

Address	Code	Label	Mnemonic	Comments

Your Machine Language Program (Continued)

Address	Code	Label	Mnemonic	Comments

Guide for Problem 1

The General Outline. The general outline for this problem is fairly simple and straightforward. You will have to initialize the data for the green light and output it to a port. Then get a 30-second time delay. Turn off the green port, initialize and output the yellow port, and get the 10-second time delay. Third, you will have to turn off the yellow port, initialize and output the red port, and get the 20-second time delay. Finally, you have to repeat the entire process.

The outline could be as follows:

1) Data for green on.
2) Output to port.
3) Get 30-second delay.
4) After 30 seconds, turn off green.
5) Data for yellow on.
6) Output to port.
7) Get 10-second delay.
8) After 10 seconds, turn off yellow.
9) Data for red on.
10) Output to port.
11) Get 20-second delay.
12) After 20 seconds, turn off red.
13) Loop.

The Refined Outline. The refined outline should identify the parts of the machine you will use in each step. For example:

1. Data will be moved to A and should be 377_8 to light all lamps for a specific port.
2. The ports should be Port 2 = green, Port 0 = yellow, and Port 1 = red.
3. You will need three time-delay routines.
4. Data will be moved to A and should be 000_8 to turn the lamps in that port off.

The refined outline should look something like:

1. Load 377_8 into A.
2. Output A to Port 2.
3. Get 30-second delay.
   a. Load delay loop count into B.
   b. Load 2 counter bytes into RP D-E.
   c. Decrement count by 1.

- d. Move upper byte from D to A.
- e. OR A and E.
- f. If result equals zero, continue; if not equal zero, go back to 3-C.
- g. Decrement B by 1.
- h. Move zeros into A.
- i. Compare B and A.
- j. Equal continue; not equal, go back to 3-b.
- k. Return to main program.

4. Load 000_8 into A.
5. Output A to Port 2.
6. Load 377_8 into A.
7. Output A to Port 0.
8. Get 10-second delay (same as 3 except 3-a small).
9. Load 000_8 into A.
10. Output A to Port 0.
11. Load 377_8 into A.
12. Output A to Port 1.
13. Get 20-second delay (same as 3 except 3-a slightly smaller).
14. Load 000_8 into A.
15. Output A to Port 1.
16. Jump back to 1.

It should be noted that this program could be one continuous program listing or could contain three **CALL** statements to get the time delays. Since it is usually easier to debug short routines, you should use the **CALL** statements.

The Flowchart. The following basic flowcharts shown in figures 2 and 3 represent one possible method to solve this problem. Note the new symbol that stands for a predefined process. The time delay flowchart is for a basic timer, and the data for each time delay cycle will have to change.

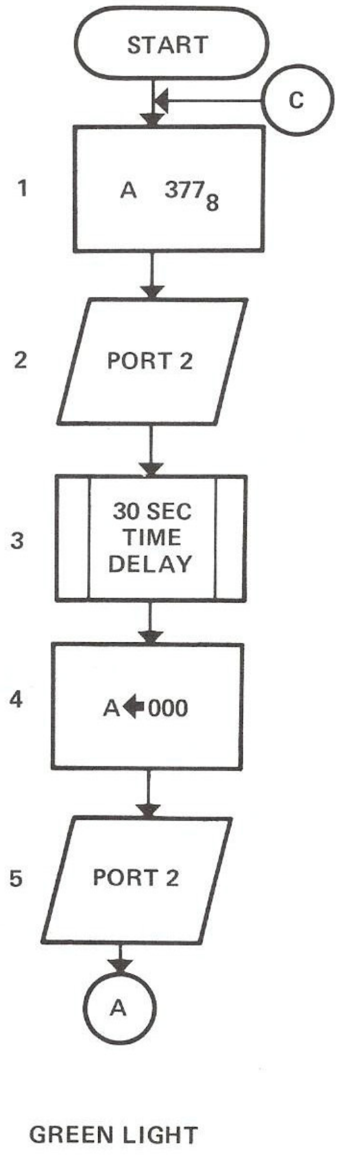


Figure 2. Basic Flowchart

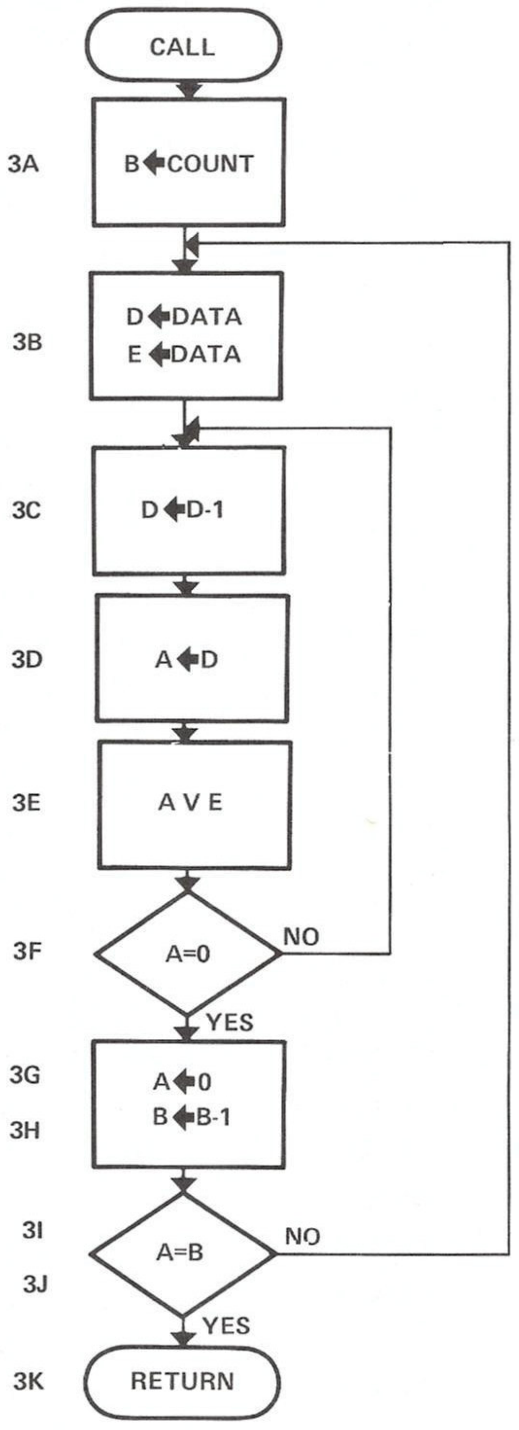


Figure 3. Basic Timer

Now you can start to code the flowchart into Assembly language. This step should use the symbolic addressing at the present time. This section contains the lamp routines, with the appropriate **CALL** statement for the time delay subroutine. The time delay subroutines will follow this section.

MAIN PROGRAM

Step	Label	Operand	Arguments	Comments
1.	START	**MVI A**	377	377 in A.
2.		**OUT**	002	Output Port 2.
3.	30-sec D	**CALL**		Get 30-sec time delay.
4.		**MVI A**	000	Zero in A.
5.		**OUT**	002	Output Port 2.
6.		**MVI A**	377	377 in A.
7.		**OUT**	000	Output Port 0.
8.	10-sec D	**CALL**		Get 10-sec time delay.
9.		**MVI A**	000	Zero in A.
10.		**OUT**	000	Output Port 0.
11.		**MVI A**	377	377 in A.
12.		**OUT**	001	Output Port 1.
13.	20-sec D	**CALL**		Get 20-sec time delay.
14.		**MVI A**	000	Zero in A.
15.		**OUT**	001	Output Port 1.
16.		**JMP** START		Go back.

BASIC TIME DELAY

Step	Label	Operand	Arguments	Comments
3a	START	**MVI B**	NNN	Count in B.
3b	LOOP 1	**LXI D**	377 377	Count in D-E.
3c	LOOP 2	**DCX D**		D-E minus 1.
3d		**MOV A, D**		Move D to A.
3e		**ORA E**		OR it with E.
3f		**JNZ**	LOOP 2	Not zero, go back.
3g		**DCR B**		B minus 1.
3h		**MVI A**	000	Zero in A.
3i		**CMP B**		Compare B and A.
3j		**JNZ**	LOOP 1	Not zero, go back.
3k		**RET**		Finished, get main program.

As stated, this is the basic time delay routine. The argument field for 3a (NNN) will vary, depending on the number of total loops required for each time delay. To help you figure this delay, the program listed requires about 1.7 seconds to run from 3b through 3k. You will have to supply the number of times (NNN) this program will have to run.

The Symbol Table and Machine Language Program. All of the required information has been provided in the Assembly language listings for you to complete these two sections yourself. Therefore, you can assign the required addresses for the symbol table and then list the instructions and codes in order. After you have completed the programming, enter and execute your program.

Summary

In this lab you have applied your knowledge about programming to solve a problem statement. You generated the general outline of the problem and refined this outline to state all the conditions. Then you developed a flowchart for the refined outline. From this flowchart, you were able to generate a simple Assembly language program. Finally you coded the program, assigned address locations to the instructions, entered the program, using the trainer, and executed the program.

The Assembled CPU Lab

Purpose

This lab provides you with the opportunity to identify microprocessor support circuits. This identification includes chip-pin number location, circuit connection, and function within the system.

Objectives

- To identify circuit components
- To identify circuit connections
- To identify circuit functions

Equipment

- MMD-2 Mini-Micro Design/Trainer

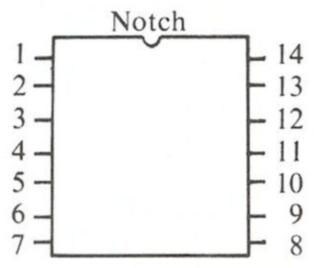
Introduction

By now you should be familiar with several microprocessor support circuits and the 8080 microprocessor itself; but do you know how these individual components combine to form a microcomputer, how to decide which integrated circuit (IC) pin is number 1, or what the schematic diagram of a microcomputer looks like? These and other questions will be answered in this lab. The schematic diagrams appearing in this lab are reprinted with permission from E & L Instruments, Inc.

Procedure

All integrated circuits will have either a notch located at one end with pin 1 the first pin immediately to the left of this notch, or, in addition to the notch, some ICs will have a small circular indentation next to pin 1. All ICs on your MMD-2 have notches at one end.

The pins are numbered in a counterclockwise system. For example, a typical 14-pin IC would have the pins numbered in the following manner:



Step 1. Locate and identify the following pins on the 8080 as shown on the *CPU and MEMORY* schematic diagram (figure 4). Compare your answer with the Answer Key at the end of this lab.

NOTE: The notch and numbering method apply to the actual IC and will not apply to schematic diagrams.

1. Address 1_____
2. Address 5_____
3. Address 15_____
4. Data 1_____
5. Data 5_____
6. Data 7_____
7. $\overline{\text{WR}}$_____
8. + 5v_____

Step 2. Locate the 8228/38 (IC15) on the *CPU and MEMORY* schematic diagram (figure 4). This is the SYSTEM CONTROLLER for the 8080. Now locate and identify the following pins on this IC. Compare your answers with the Answer Key at the end of this lab.

1. $\overline{\text{Memory Read}}$_____
2. Data Bus In _____
3. Bidirectional Data Bit 4_____
4. Hold Acknowledge _____
5. System Status Strobe_____

Question. The System Status Strobe is generated by IC____________, pin ____________. This IC is an____________ and its function is ____. (14, 7, 8224, clock generator)

Step 3. Locate ICs 24 and 26. These are the 1K x 4 read/write memories.

Question. Which pin is the Write Enable signal?________________________ (10 on both)

Question. Which IC contains the low-order nibble (bits 0 through 3)?____________(26)

Step 4. Locate ICs 16 and 17 on the *CPU and MEMORY* schematic (figure 4).

Question. What are the functions of these two ICs?

____ (address bus drivers/buffers)

Step 5. Look at the *KEYBOARD/DISPLAY CIRCUIT* schematic diagram (figure 5).

Question. IC 11, pin 6 is active. Which display is enabled?____________________(D6)

Step 6. On the top right side of the schematic (figure 5) for the *KEYBOARD/DISPLAY CIRCUIT* is the pin number and alpha designation for each section of the seven-segment display. The numbers in parenthesis are the pin numbers for that segment of the display.

Question. IC 12 has the following pins active: 1, 2, 9, 12, and 14. What is the character being displayed?______________________________(E)

Step 7. On the MMD-2 trainer, locate the following ICs:

1. 8080
2. 2708
3. 2111A
4. 2114
5. 8228/38
6. 8279
7. 74LS74
8. CA3081

When you locate each of these ICs, you should note the physical characteristics of each of these devices. These should include the size, number of pins, and type of construction material (plastic or ceramic).

Note that ceramic is a much better heat conductor than plastic. Therefore, on certain ICs that require more stability or that perform a greater amount of work, the construction may be of ceramic.

You should become familiar with these types of configurations to help you when working with future systems.

Study the schematic diagrams shown in figures 4 and 5 and see if you can trace the various connections on the printed circuit board (the MMD-2).

In general, become familiar with the drawings, assembly configurations, and functions of the various circuits.

See if you can trace the operation and flow of various signals.

Summary

In this lab, you were shown partial schematic diagrams of the MMD-2 trainer. These are typical schematics showing the various circuit symbols, connections, and operational characteristics for a specific microprocessor system. You also saw or noted the size, shape, and construction of various ICs including the 8080, ROM, RAM, and system controller.

In this lab, you identified various circuit components, connections, and functions. You were able to locate pin 1 of various ICs, circuit connections for address and data lines, and the functions of various circuits.

In this lab, you learned the operation of various circuits by tracing the circuit data flow. You learned how to identify pin 1 on these ICs. You also learned how the various circuits are connected to form a microcomputer.

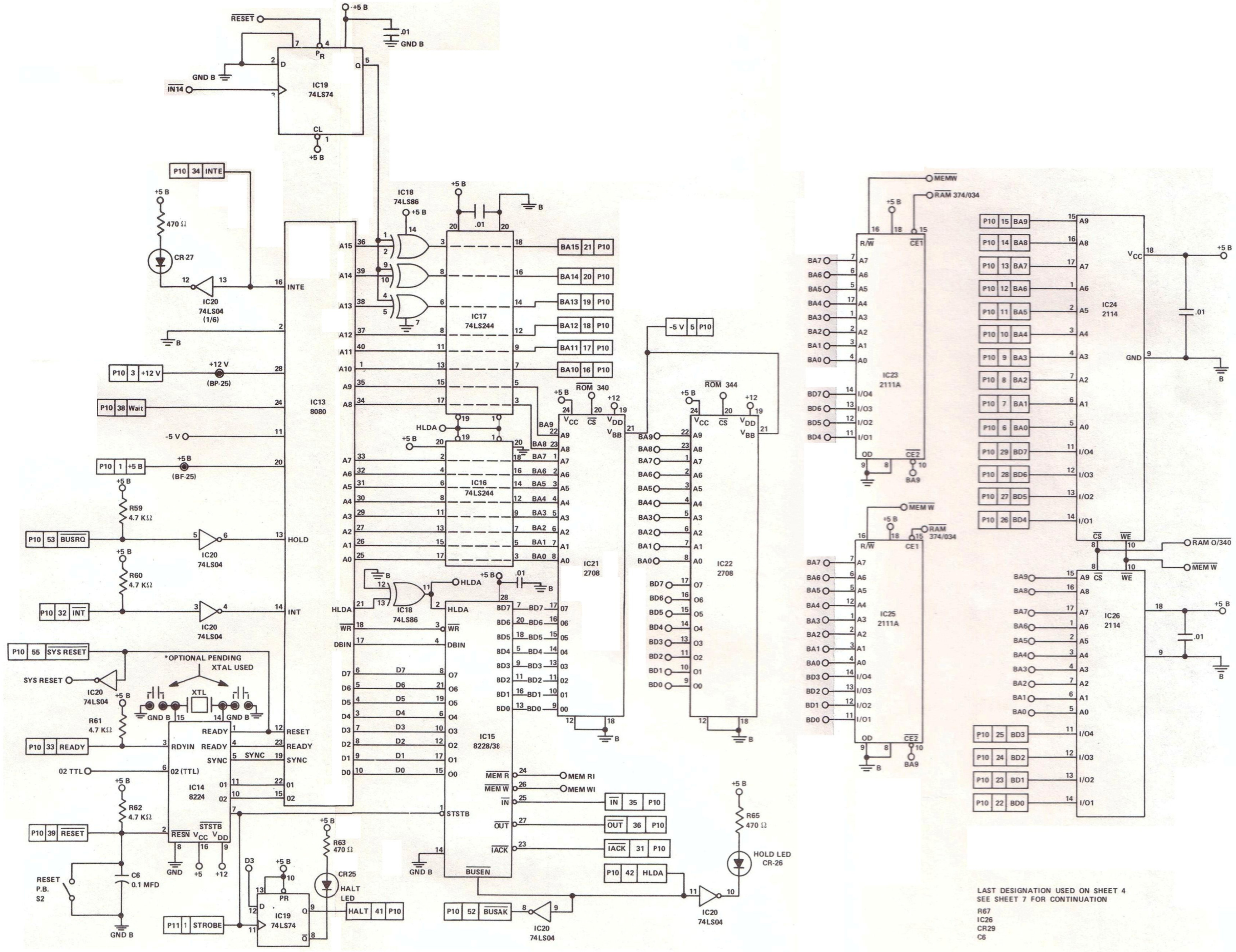


Figure 4. CPU and Memory Schematic Diagram

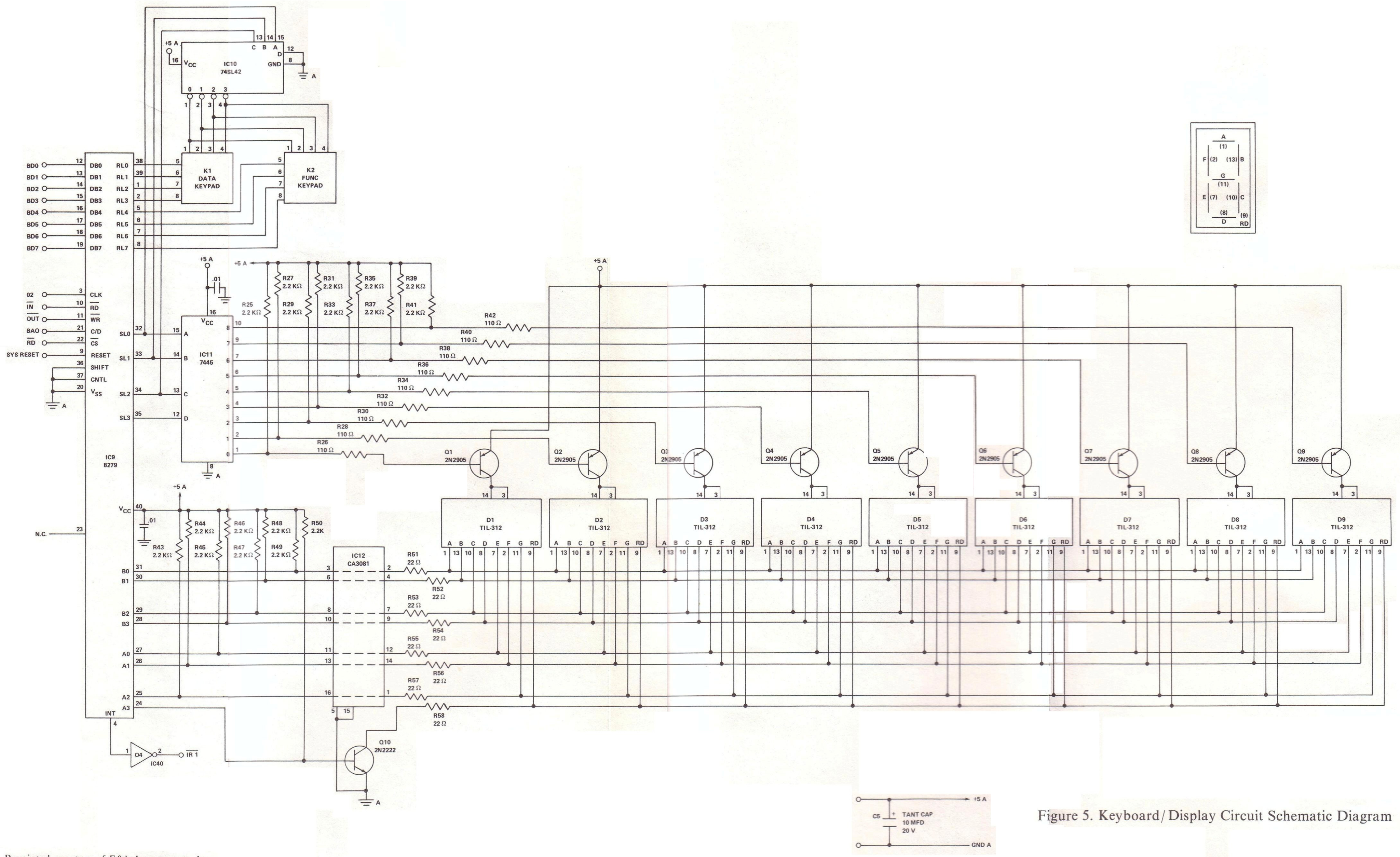


Figure 5. Keyboard/Display Circuit Schematic Diagram

Reprinted courtesy of E&L Instruments, Inc.

Answer Key

Step 1.

1. 26
2. 31
3. 36
4. 9
5. 4
6. 6
7. 18
8. 20

Step 2.

1. 24
2. 4
3. 5
4. 2
5. 1

D/A Conversion Lab

Purpose

This lab will have two purposes. The first will be to introduce you to the digital-to-analog circuits and operation, as configured on the Experimental Interface Designer (EID-1). The second will be to enable you to program this typical digital-to-analog converter.

Objectives

- To identify the functional operation of a D/A converter
- To generate, enter, and execute a simple D/A conversion program

Equipment

- MMD-2 Mini-Micro Design/Trainer
- EID-1 Experiment Interface Designer
- MMD-2 to EID-1 Interface Ribbon Cable

Introduction

The Experimental Interface Designer (EID-1) contains the required circuitry for you to write programs for both D/A and A/D conversion processes.

The D/A circuits consist of a D/A converter and an analog DC meter representing one type of loading device. A DC motor is also supplied as a second loading device. Since this motor is DC-operated, its speed will depend on the amount of voltage applied to it. Therefore, it is a variable speed motor. (See figures 9 and 10.)

The A/D portion (used in following labs) consists of a photocell, a temperature sense circuit, a linear slidepot, and a hardware-constructed A/D converter similar to those previously studied. The photocell, temperature sense, and slidepot are independent analog inputs that are switch selectable (S2). (See figures 11 through 13.)

The EID-1 also has an eight-position DIP (Dual In-line Package) rocker switch that is connected to the I/O Bus and an output port with red, yellow, and green LEDs as displays. The red LEDs are the two most significant bits, and the green LEDs are the first three least significant bits. (See figure 7.) Each circuit (A/D, D/A, input, and output) is independent of each other. The interconnection of the circuits is controlled by the programs you are about to develop.

Digital-to-analog converters, as you should know, are devices that will convert (change) a digital (on/off) input signal into an analog (variable) output signal that can control some process. This process could include speed or position control. This lab allows you to develop a program to control the rotational speed of the DC motor.

Procedure

The locations of the analog devices and controls you are about to use are as follows: (locate each as it is described)

In the bottom center of the EID-1 is the eight position input DIP switch labeled S3. (The switch numbered 1 is the MSB.) To the right of this switch (in the lower right corner) is the position sensor. This is labeled R39 and is the slidepot. The solder connections for the slidepot are located above it and are labeled GND, WIPER, and + 5V. Now, look at the device located just above the solder joint labeled WIPER. This device is labeled Alternate Light Devices and is the photocell. Directly above the photocell is a black plastic transistor labeled Q3. This is the temperature sensing device.

Now, to the right of the photocell is a four position slide switch labeled S2. This switch is located along the right edge of the EID-1 and selects the analog input device. The top position selects the temperature sensor, the second position down from the top selects the photocell, the third position selects the position sensor, and the fourth position "opens" the analog section. When the fourth position (bottom) is selected, the meter and motor will indicate maximum conditions. This position should NOT be used during any of these labs.

Located just above the device select switch (S2) is another switch labeled S1. This is the motor on/off switch. When the switch is in the left position, the motor is on.

Now, perform the following steps to connect the EID-1 and perform the lab experiment.

Step 1. Interface Connection

WARNING: When connecting or disconnecting the MMD-2 and EID-1 together, the following practices MUST be observed to prevent damage to the equipment, the circuits, and yourself.

1. *MMD-2 Power must be OFF.*
2. *CARE must be taken when CONNECTING or DISCONNECTING interface cable connectors.*

The interface cable has a 40-pin receptacle (female) connector that plugs into the 40-pin plug (male) connector mounted on the EID-1.

The other end of the ribbon cable also has a 40-pin connector that connects to a printed circuit card. This PC card is called a "paddle card" and contains the circuits for generating several required STD bus interface signals.

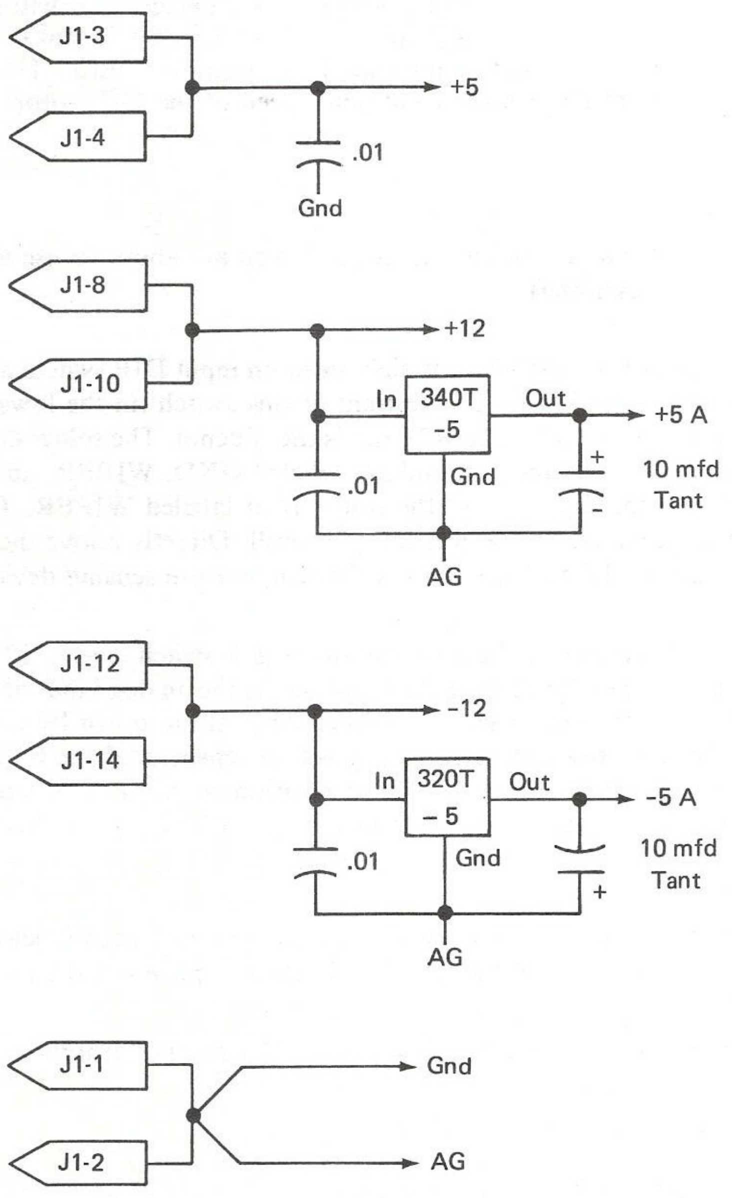


Figure 6. EID-1 Power Connections Schematic Diagrams

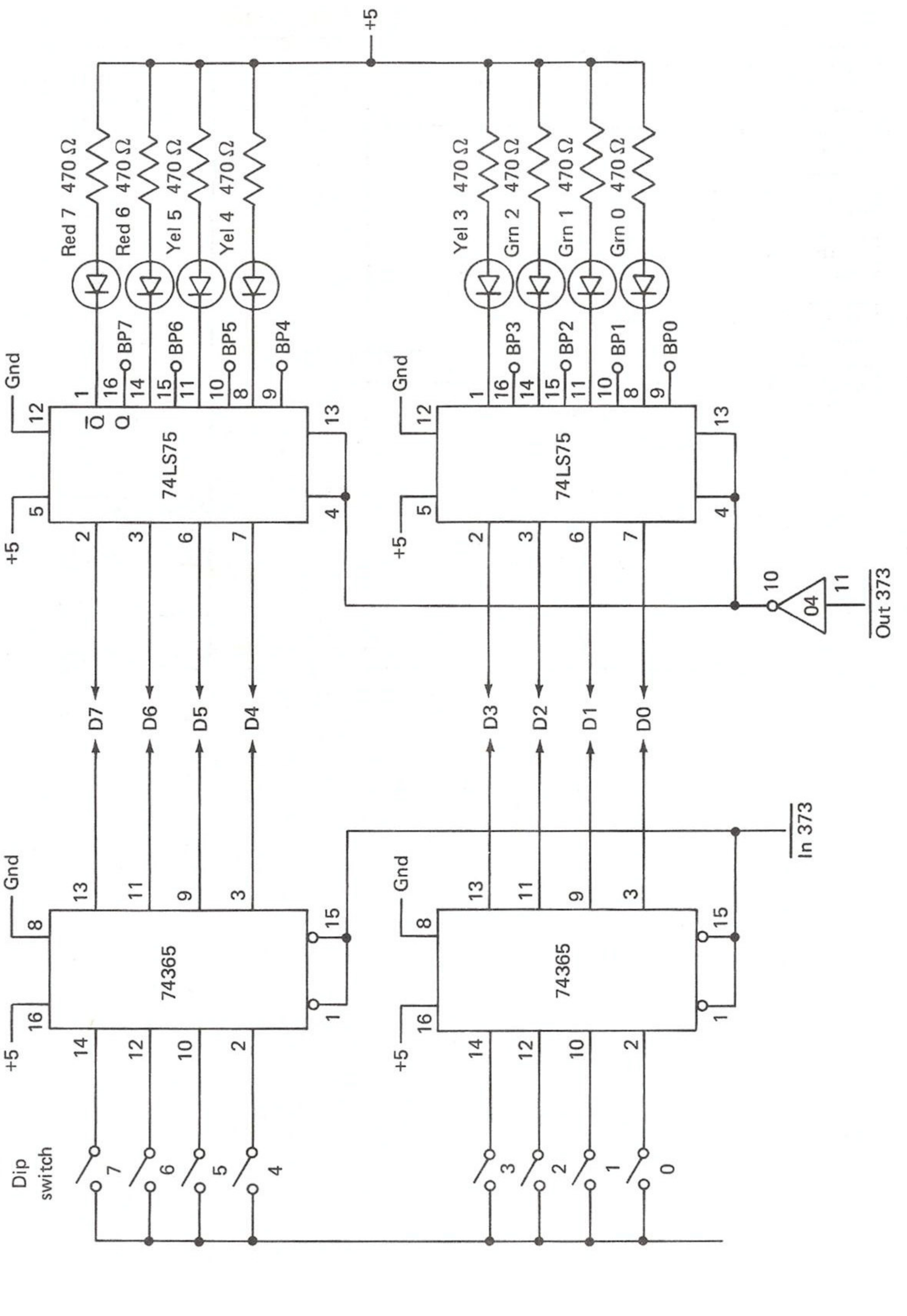


Figure 7. I/O Ports

Reprinted courtesy of E&L Instruments, Inc.

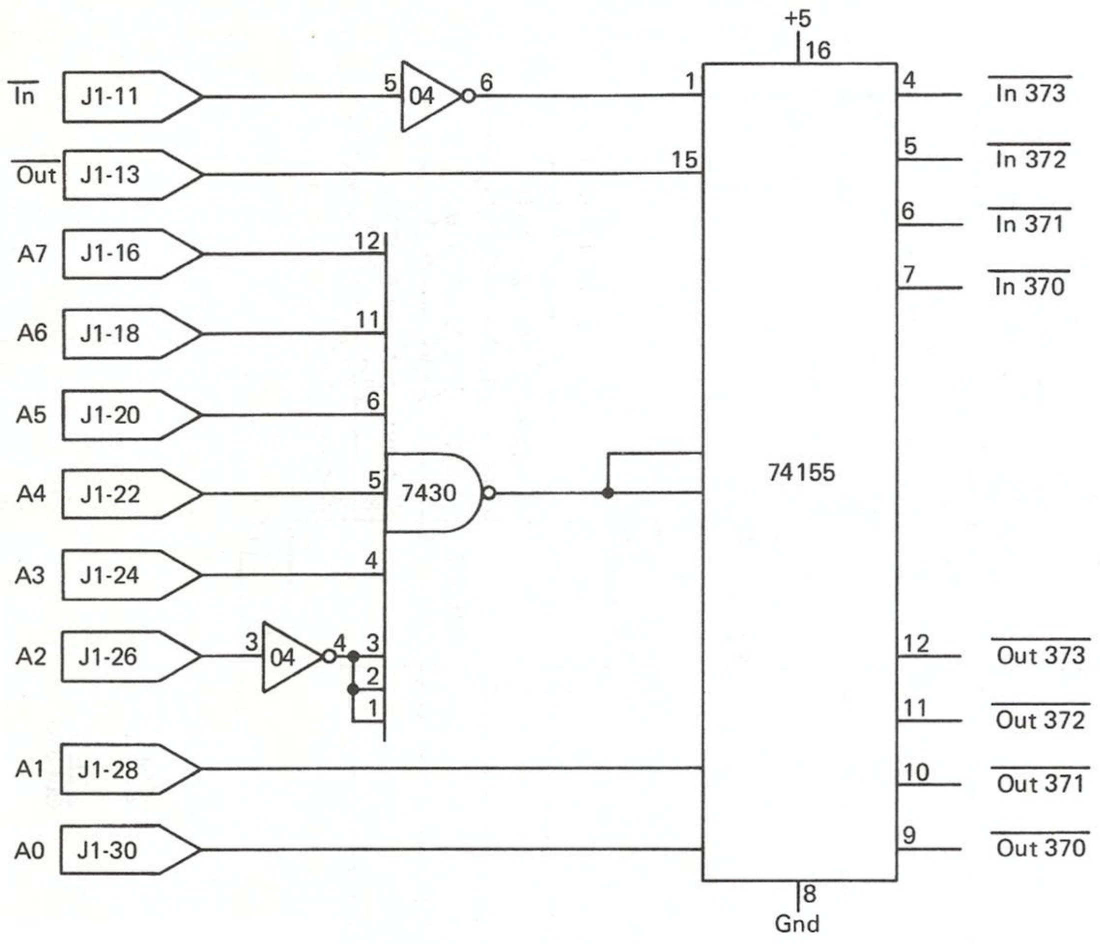


Figure 8. EID-1 Device Decoding

Reprinted courtesy of E&I. Instruments, Inc.

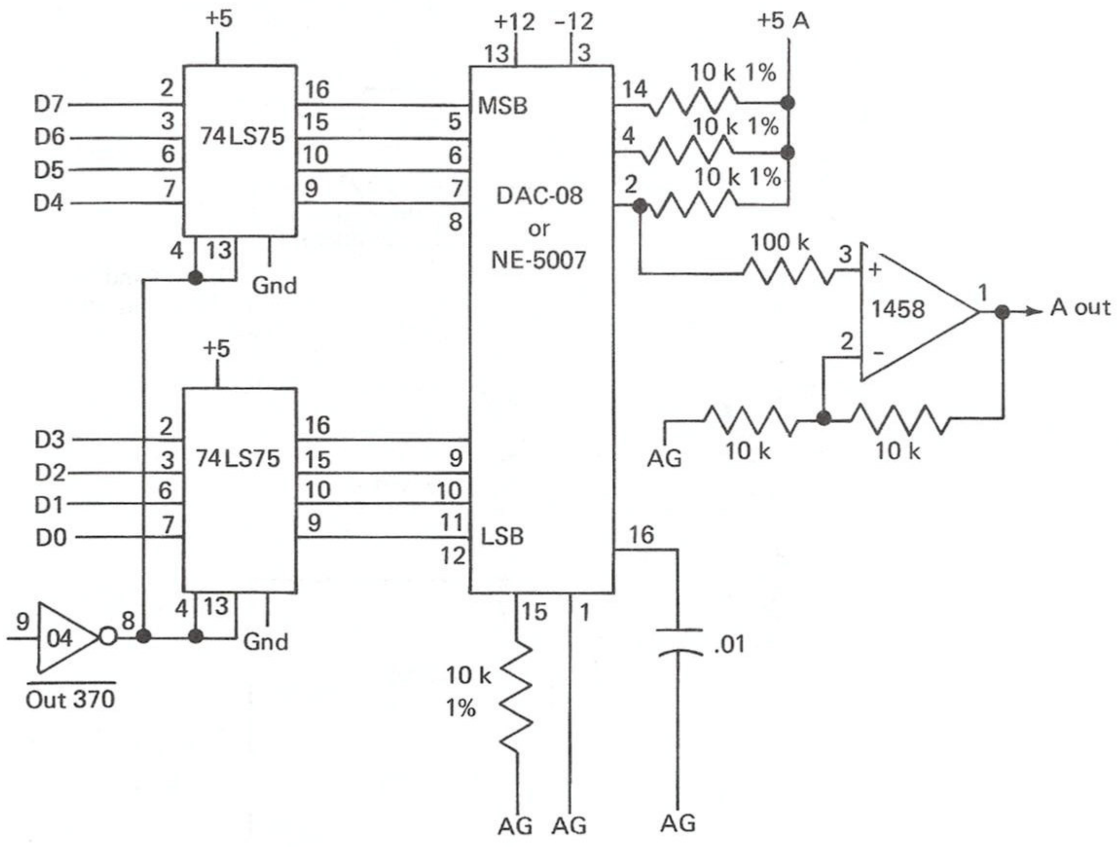


Figure 9. EID-1 D/A Converter

Reprinted courtesy of E&L Instruments, Inc.

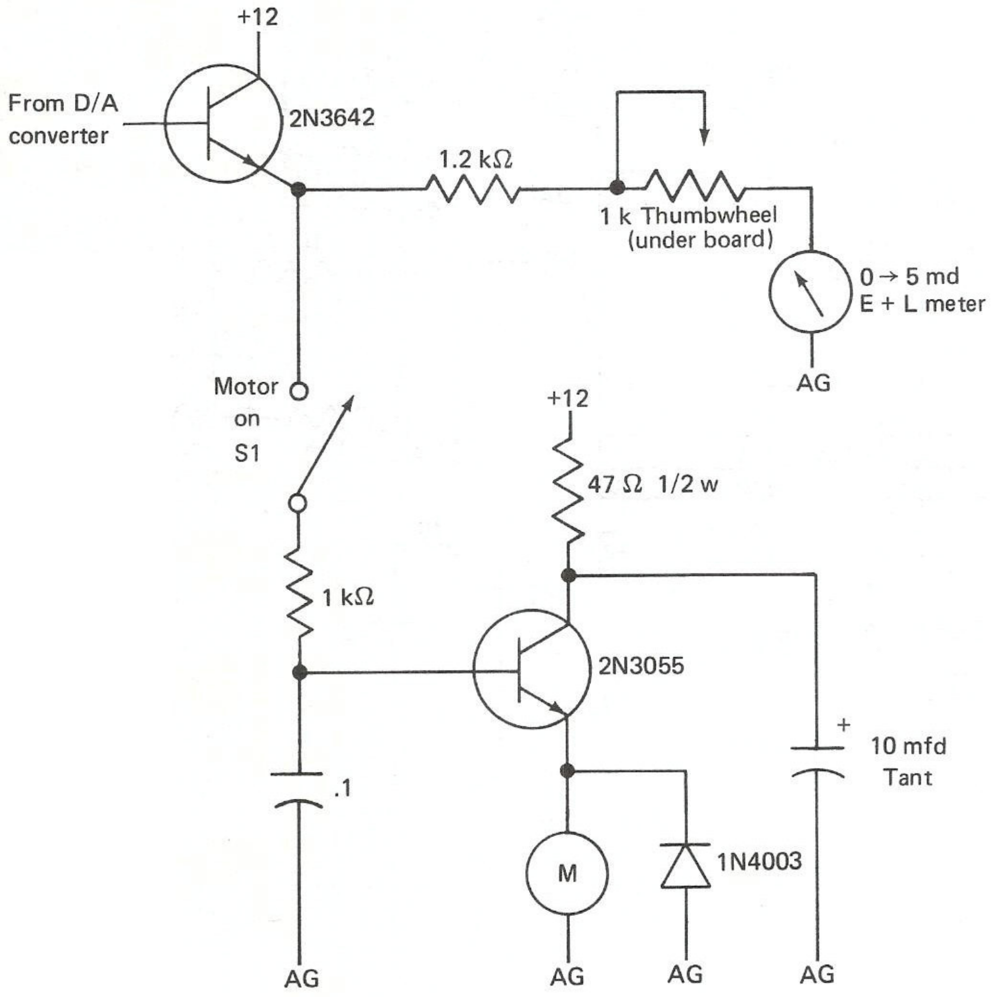
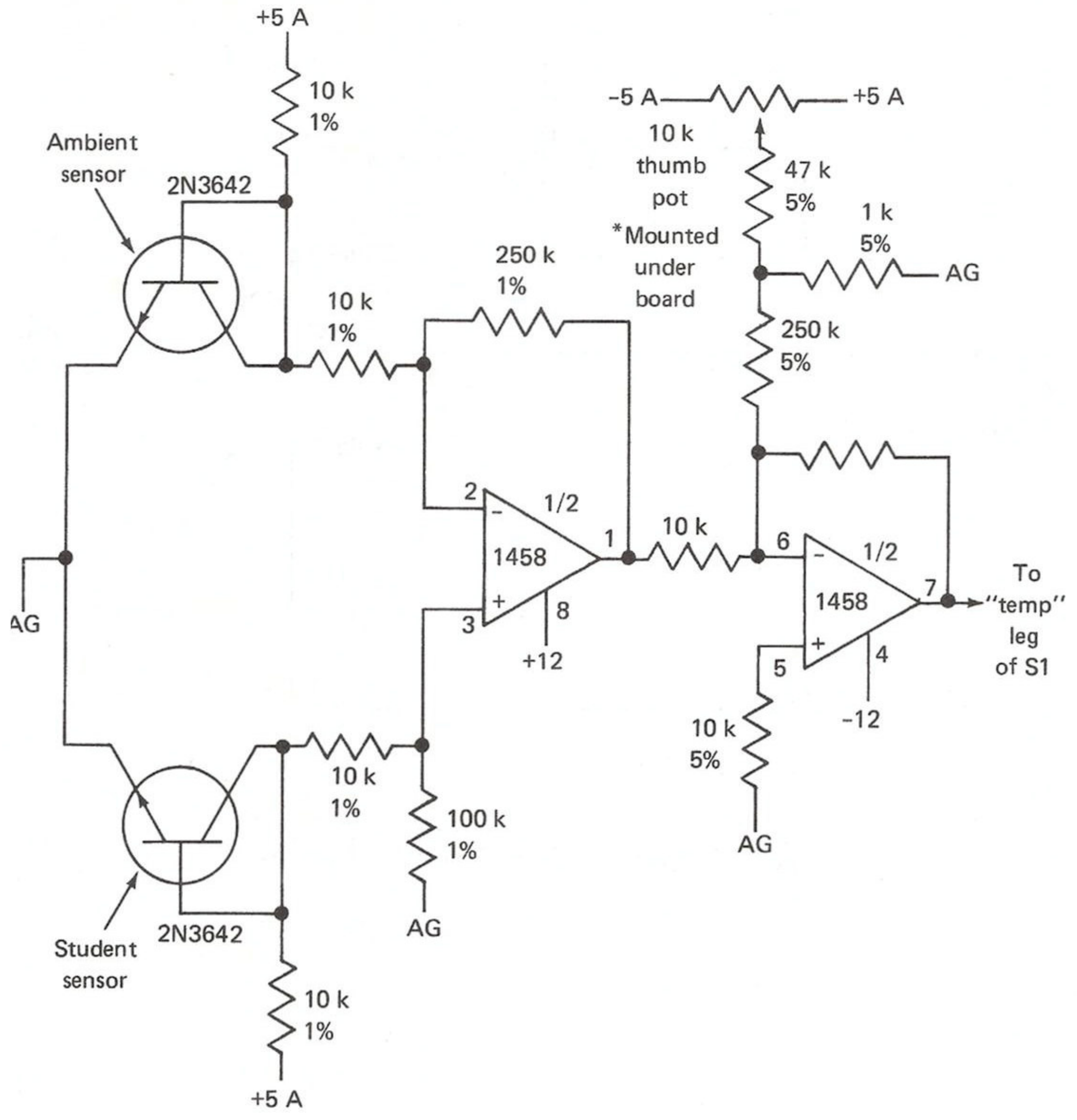


Figure 10. EID-1 Motor and Meter Circuit



Calibration
1) Turn on
2) Warm up 5 min
3) Adj pot to yield +2.0 VDC at "temp" leg of S1
4) Seal pot with glyptol

Figure 11. EID-1 Temperature Circuit

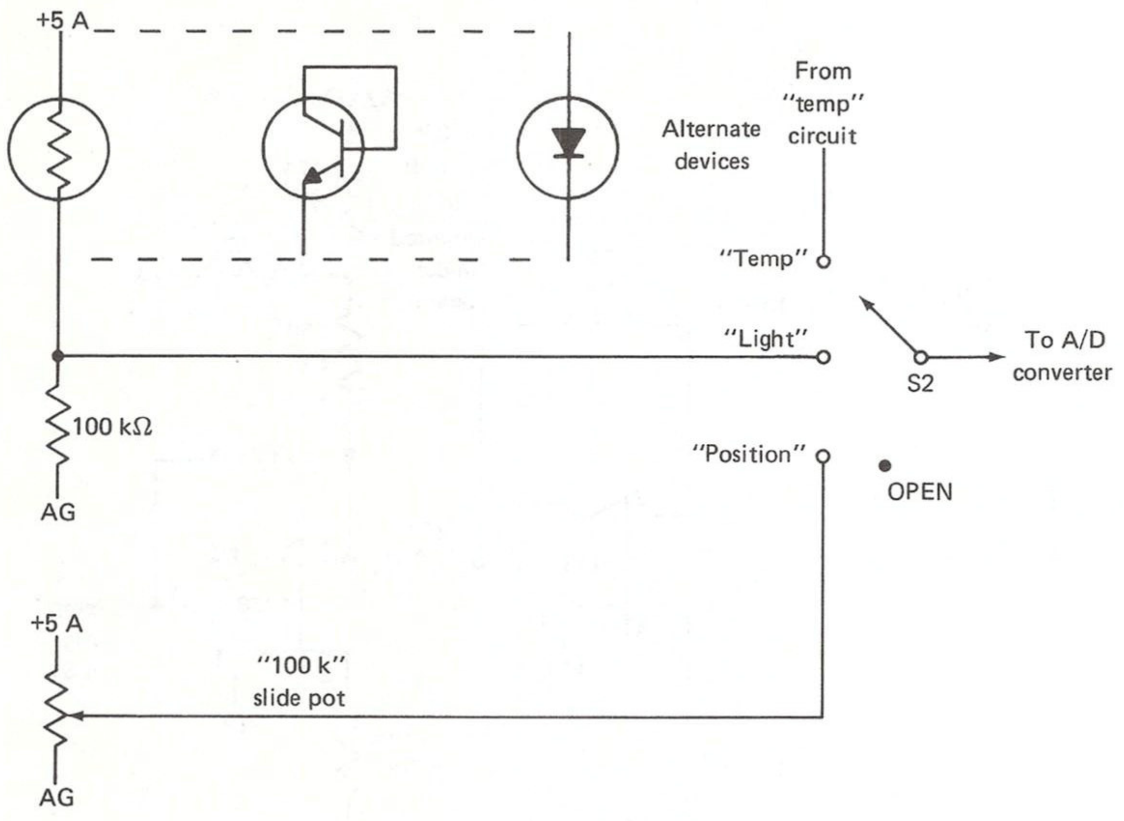


Figure 12. EID-1 Light and Position Circuit

Reprinted courtesy of E&L Instruments, Inc.

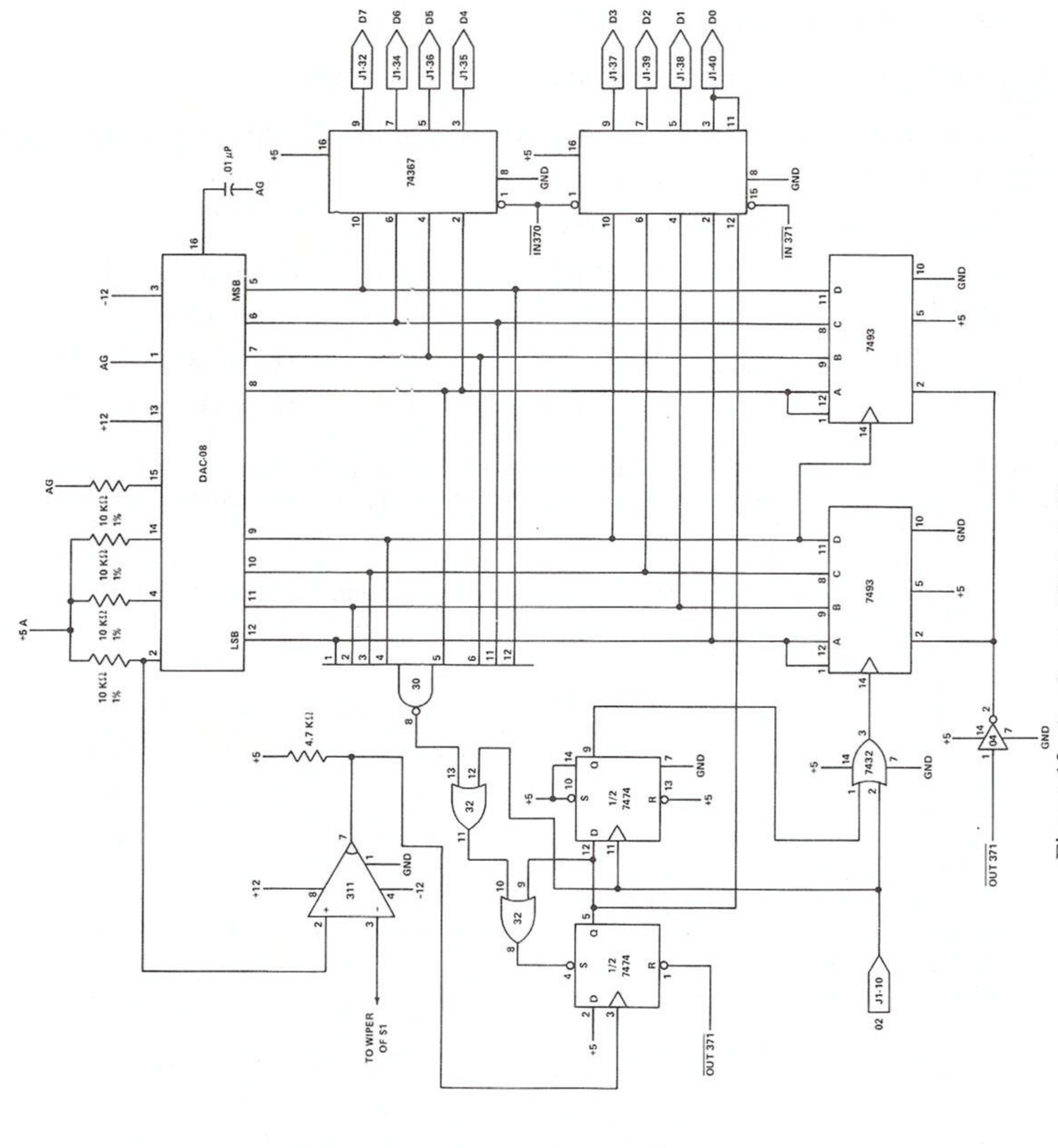


Figure 13. Analog to Digital Circuits

Reprinted courtesy of E&I. Instruments, Inc.

When connecting the ribbon cable connectors, be sure the pin alignment (pin 1 to pin 1) and all others are correct to avoid bending the connecting pings. Pin 1 has a small arrow molded on the plastic connector.

The paddle card also has a 56-pin card edge connector (male) that connects to the MMD-2. The MMD-2's 56-pin connector is located on the left-side chassis. Again, be sure of the pin alignment before insertion.

Step 2. After you have verified the interface connections, apply power to the MMD-2.

Step 3. Write a program that will continuously input from the DIP switches and output to the port.

The input and output port address is 373.

	Code	Mnemonic	Comment
LOOP	___ ___	**IN**	Input 373.
	___ ___	**OUT**	Output 373.
	___ ___ ___	**JMP** LOOP	Go back.

This program will output to the LEDs the bit positions representing the switch setting. Execute the program. As you SET and RESET the various input switches, verify the corresponding output LED is lit.

Step 4. Turn the MOTOR switch off. Modify your program to output to the D/A converter. The program should now look like:

Address	Code	Mnemonic	Comment
003000	333	**IN**	Input 373.
003001	373	373	
003002	323	**OUT**	Output 373.
003003	373	373	
003004	323	**OUT**	Output 370.
003005	370	370	
003006	303	**JMP**	Go back to 003000.
003007	000	000	
003010	003	003	

Step 5. Place all of the DIP switches in the OFF (zero) position.

The eight DIP switches represent an 8-bit binary number. When the program is executing, it runs in an endless loop and continually reads in the binary number on the switches; then it displays it on the eight LEDs on the interface board; and finally, it outputs the binary number to the D/A converter. The analog output of the converter is measured by the small voltmeter on the interface board.

The leftmost bit of the binary number and the DIP switch is the *most significant bit*. When it is on, it represents 5.0 volts. The next bit to the right is the next most significant bit, and it represents 2.5 volts, or exactly one-half of the value of the most significant bit. Each bit to the right represents a voltage that is one-half its left neighbor. The following table shows what voltage each switch or bit position represents:

Switch or Bit	Voltage (in volts)
8	5
7	2.5
6	1.25
5	0.62
4	0.3125
3	0.15625
2	0.078125
1	0.0390625

Starting with the least significant bit, each succeeding position is twice as large as the previous one. This is exactly like binary coding except that the least significant bit, instead of representing a one, represents 0.0390625 volts.

Now suppose that switches 1 through 6 were all set to zeros, and switches 7 and 8 were set to ones.

Question. What voltage would you expect to find on the meter?

____________________________________ (7.5 volts)

Step 6. Set switches 1 through 6 to zero and switches 7 and 8 to 1.

Question. What do you observe on the meter?

____________________________________ (7.5 volts)

Just as in a binary number, each bit position represents a binary weight to be added "in" to form a resulting number, each position for the D/A converter represents a voltage weight that will be summed by the converter to form the resulting analog voltage. Since only switches 7 and 8 are ones in this case, the voltage weights that these switches represent (i.e., 5.0 volts for switch 8 and 2.5 volts for switch 7) will be summed to get the final result, 5.0 + 2.5 = 7.5 volts.

Step 7. Try the different combinations of switches shown below.

	Switches 8 7 6 5 4 3 2 1	Voltage
A)	1 0 1 0 0 0 0 0	__________
B)	0 1 1 0 0 0 0 0	__________
C)	0 1 1 0 0 0 0 1	__________
D)	1 1 1 1 1 1 1 1	__________

Question. What was the difference between B and C?

____

(B and C are only 0.0390625 volts apart, and we could not observe this small difference on our meter.)

Question. What was D? ________________________________

____

(Actually, D is the sum of all possible voltage weights, or

$$5.0 + 2.5 + 1.25 + .625 + .3125 + .15625 + .078125 + .0390625$$

which all adds up to 9.9609375 volts. The sum is so close to 10.0 volts that you can't detect the difference on our meter.)

Question. Can you see how to generate a voltage that is as close as possible to any voltage between 0.0 and 10.0 volts? ________________

(Yes, by setting the various combinations of the switches, you can generate almost any voltage between 0.0 and 10.0 volts.)

Step 8. Generate the voltage that is closest to 4.0 volts.

Question. Which switches should be set to ones and which should be zeros?

Switches

8 7 6 5 4 3 2 1

(Switches 7, 6, 3 and 2 to one and all others to zero. This adds up to 3.984375 volts. If we wanted a voltage a little larger than 4.0 volts, we could have made switch 1 also a one. That would have given us 4.0234375 volts which is not as close to 4.0 volts as 3.984375. With just switches 7, 6 and 4 as ones and all the rest zeros, we would get 4.0625 which is still further away from 4.0 volts, but not by much.)

Step 9. Set your switches to generate the following voltages:

Voltage	Switches 8 7 6 5 4 3 2 1
3.75	__________
1.0	__________
2.6	__________
3.0	__________
6.5	__________
7.0	__________
8.0	__________
9.5	__________

Step 10. Turn the MOTOR switch on.

Set the DIP switches to read 377_8 (all ones).

Record the meter reading.

This reading may be slightly lower than the 10 volts previously read due to the current losses in the windings of the motor. However, the motor is buffered, which keeps most of this loading from affecting the circuit.

Step 11. Set all the DIP switches to zero. With the MOTOR switch ON, gradually set the switches until the motor starts to run. Record the value as read on the EID-1 meter.____________Try it several times to verify your reading. The value should be approximately 2.5 vdc.

Then, with the motor running, gradually decrease the voltage applied by resetting the switches until the motor stops. Record the value as read on the EID-1 meter. ____________Try it several times to verify your reading. The value should be approximately 2.0 vdc.

What you found in the first part of this step was the initial start-up voltage of the motor. This is the "force" required to overcome the stationary conditions of the motor. This stationary condition is called *inertia*. As you can see in the second part of this step, once inertia has been overcome, the motor will run at a lower voltage.

Remember, this motor does not have a load on it; therefore, the inertia is lower. If the motor were attempting to drive something, the inertia of this load would also have to be considered.

Step 12. Modify the first two address locations of your program to read

```
076     MVI A
200
```

Execute your program.

Question. What does the meter read now?________________________(5 volts)

Step 13. Modify address location 003001 to result in a voltage of 10.

Execute the program.

Question. What is the value in A?____________________________(377)

You have now used the microprocessor to control the output of an analog device, using two methods. The first was a switch input to set the bit on the output device. The second method (steps 12 and 13) loaded the A register with a digital value and output this value to the D/A converter.

As you remember, one of the functions of a D/A converter could be to control the speed of a motor. For example, the fluid control during a chemical process may require the pump to gradually increase the flow until the maximum flow is reached. When the required amount of fluid has been reached, the pump must gradually decrease until it is shut off, or it may be required to maintain a minimum flow.

Step 14. Write a program that will monitor the LSB on the DIP switch. When the switch is turned on, the microprocessor will control the motor speed by slowly increasing the speed until a specified maximum output (speed) is reached. It will maintain this maximum output until the switch is turned off. When the switch is turned off, the motor speed will slowly decrease until a specified minimum speed is reached. It will maintain this minimum output speed until the switch is toggled on again.

The flowchart (figure 14) on the next page represents the program you should write. The following guide will help you to write this program. Finally, the entire program is listed at the end of this guide.

The Guide

The first part of your program should be the *switch monitor* routine. This section should input the switch input port and look at the settings. If they are correct start the main program. If not correct, go back and look again.

Step 15. Write this routine and compare it with the program listing, *or* complete the comment section of the program listing. The program listing is located at the end of this lab.

Address	Code	Label	Mnemonic	Comment

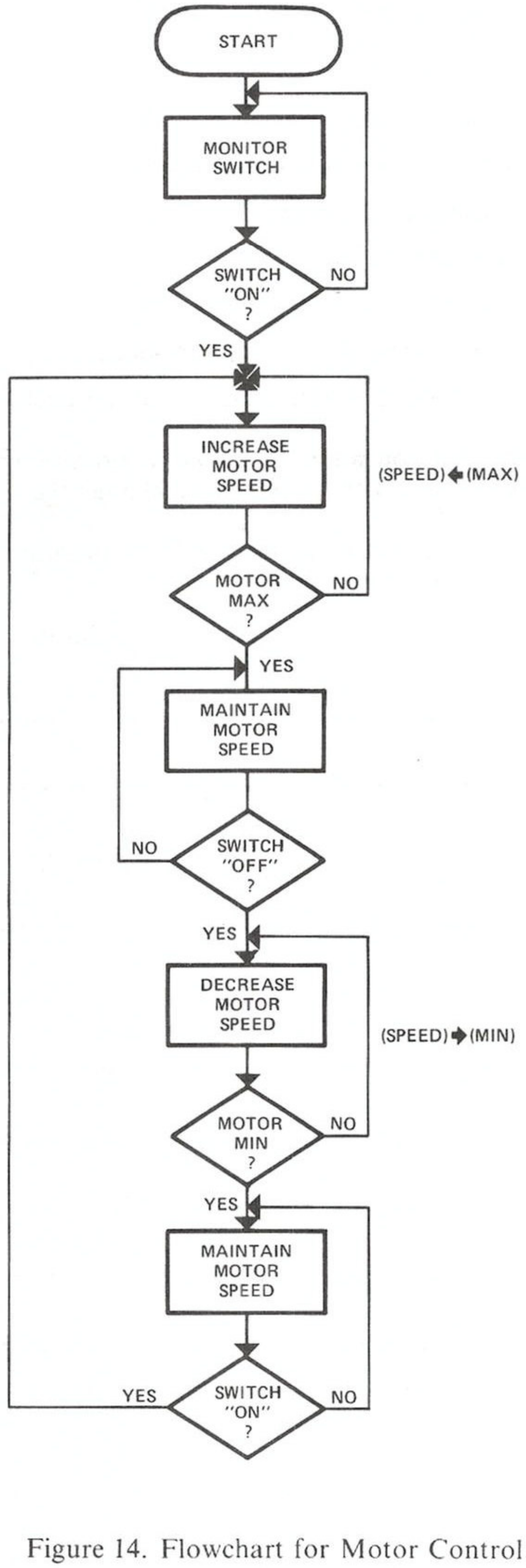


Figure 14. Flowchart for Motor Control

The second part of your program should be the *motor on—increase speed* routine. The following conditions should be met:

- Maximum speed count to a register
- Minimum speed count to another register
- Start increasing speed count
- Output the speed
- Get a time delay routine to control speed rate
- Compare present speed to maximum speed
- If equal, go monitor switch again; if not, go back and increase speed.

You should also note that you want to be able to alter the speed count so you can reach maximum speed faster without having to change the time delay.

Step 16. Write this routine and compare it with the program listing, *or* complete the comment section of the program listing.

Address	Code	Label	Mnemonic	Comment

Address	Code	Label	Mnemonic	Comment

The third part of your program should now check to determine if the switch has been turned off. If it has not been turned off, continue to output maximum speed. If it has been turned off, start to decrease the speed.

Step 17. Write this routine and compare it with the program listing *or* complete the comment section of the program listing.

Address	Code	Label	Mnemonic	Comment

The fourth part of your program should be the *motor decrease* speed routine. The following conditions should be met:

- Minimum speed count to a register
- Start decreasing the speed
- Output that speed
- Get a time delay routine to control speed rate
- Compare present speed to minimum speed
- If equal to or less than minimum, go monitor switch again; if not equal, go back and decrease speed some more

Again, you want to be able to alter speed count without having to change time delay.

Step 18. Write this routine and compare with the program listing, or complete the comment section of the program listing.

Address	Code	Label	Mnemonic	Comment

Address	Code	Label	Mnemonic	Comment

The fifth part of your program should monitor the switch again to see if it has been turned on again. If it has, start the whole process over again; if it has not, maintain minimum speed.

Step 19. Write this routine and compare with the program listing, *or* complete the comment section of the program listing.

Address	Code	Label	Mnemonic	Comment

Finally, you need to write the time delay routine. This routine should save the present speed count. Load a register pair with the time count and start counting down. When the time is up, go back to the main program.

Step 20. Write this routine and compare with the program listing, *or* complete the comment section of the program listing.

Address	Code	Label	Mnemonic	Comments

Step 21. Enter and execute the program. Verify its operation. (Don't forget to set DIP switches to zeros and the MOTOR switch ON before starting program.)

Does it meet the following specifications?

1. No output until switch (DIP) is turned on?
2. Slow increase in speed until maximum?
3. If switch (DIP) is turned off as soon as motor starts, does the speed continue to increase until maximum, then to decrease speed slowly?

4. If switch (DIP) is left on, does the motor maintain maximum speed?
5. When switch (DIP) is turned off, does motor speed decrease slowly?
6. If switch (DIP) is turned on as soon as the motor starts decreasing speed, does the motor reach minimum speed before increasing again?
7. If switch (DIP) is left off, does the motor maintain the minimum speed?
8. When switch (DIP) is turned on again, does motor speed start to increase?

(Your answers should be yes to all questions.)

Now let's take a look at some of the possible applications of this program. For example, a security device, such as a combination lock, could be developed. For example, you could set the initial compare value to some code other than bit position 1 (location 3003 in the program listing at the end of the lab). Remember, by doing this, the codes would have to be the same before the motor would start. Unless you know what the code is, it will take a rather long time to set and clear all the various combinations of switch settings.

Want to try it? Here's one method you can use.

Cover the data display so you can't see the codes. Select address location 003003. Put your hand on the DATA INPUT keyboard and move your hand around slightly. This should cause some "random number" for data in. Enter that data (don't look at the display) and execute the program. Now start setting the various switch patterns to see if you can find the setting that will start the motor.

To increase the security of this code, you could start the program with another compare looking for any switch ON, and then have a time delay look for the correct code within that time; if not received, stop the microprocessor.

Other examples of applications are:

- You could increase the number to be added to the output level (location 003014). This causes the motor to "step up" to speed. Changing location 003047 causes the motor to "step down." In both cases, the motor stabilizes at each step (time delay routine) before the next step.
- You could have the motor "jump" to maximum speed (or minimum) and slowly decrease (increase) by altering the same location (3014 or 3047) or change location 003012 for "jump to max." Since the maximum speed is already in the contents of A, "jump to min" would have to be the subtract number.
- You could have the motor maintain a smooth but fast "start-up and slow down" by altering the time delay count. Another time delay would have to be written if you wanted one of these to be slow (slow start up, fast down, or vice versa).

Feel free to make any of these alterations to the program if you want. Don't hesitate to experiment.

Program Listing

Address	Code	Label	Mnemonic	Comment
003000	333	LOOP 1	**IN**	
3001	373			
3002	376		**CPI**	
3003	001			
3004	302		**JNZ** LOOP 1	
3005	000			
3006	003			
3007	006	COND	**MVI B**	
3010	301			
3011	076		**MVI A**	
3012	001			
3013	306	LOOP 2	**ADI**	
3014	001			
3015	323		**OUT**	
3016	370			
3017	315		**CALL**	
3020	350			
3021	003			
3022	270		**CMP B**	
3023	312		**JZ** NEXT TEST 1	
3024	031			
3025	003			
3026	332		**JC** LOOP 2	
3027	013			
3030	003			
3031	323	NEXT TEST 1	**OUT**	
3032	370			
3033	117		**MOV C, A**	
3034	333		**IN**	
3035	373			
3036	376		**CPI**	
3037	000			
3040	171		**MOV A, C**	
3041	302		**JNZ** NEXT TEST 1	
3042	031			
3043	003			
3044	006		**MVI B**	
3045	012			
3046	336	LOOP 3	**SBI**	
3047	001			
3050	323		**OUT**	
3051	370			

Program Listing (Continued)

Address	Code	Label	Mnemonic	Comment
3052	315		CALL	
3053	350			
3054	003			
3055	270		**CMP B**	
3056	312		**JZ** NEXT	
3057	067			
3060	003			
3061	332		**JC** NEXT TEST 2	
3062	067			
3063	003			
3064	303		**JMP** LOOP 3	
3065	046			
3066	003			
3067	323	NEXT TEST 2	**OUT**	
3070	370			
3071	117		**MOV C, A**	
3072	333		**IN**	
3073	373			
3074	376		**CPI**	
3075	001			
3076	312		**JZ** COND	
3077	007			
3100	003			
3101	171		**MOV A, C**	
3102	303		**JMP** NEXT TEST 2	
3103	067			
3104	003			

Time Delay

Address	Code	Label	Mnemonic	Comment
003350	117		**MOV C, A**	
3351	021		**LXI D**	
3352	050			
3353	010			
3354	033	COUNT	**DCX D**	
3355	172		**MOV A, D**	
3356	263		**ORA E**	
3357	302		**JNZ** COUNT	
3360	354			
3361	003			
3362	171		**MOV A, C**	
3363	311		**RET.**	

Summary

In this lab you were introduced to the Experimental Interface Designer (EID-1) trainer, which contains D/A, A/D converters, an input port (DIP switch), and another output port.

You studied the schematic diagrams to see how each of these typical devices is interfaced to a microprocessor. You also saw how each was configured within its circuit.

You started the lab experiment by doing an IN/OUT routine and saw how the switches could control the output. Then you also used these switches to output to a D/A converter. With this program, you saw how the bit setting (digital) affected the D/A output (analog); this output level was read on a meter, and a digital bit weight was found. Then you turned on the motor; by using the microprocessor to generate the various bit patterns, you saw how this affected the speed of the motor.

Then you generated, entered, and executed a program to turn the motor on when a switch was set and controlled the speed of the motor with the same program.

Finally, you were able to alter certain parameters in the program to have the motor perform in different mannerisms. Some of these mannerisms were step-up, step-down, fast up, fast down, and security on.

A/D Conversion Lab

Purpose

The purpose of this lab is to introduce you to one of the programming techniques required for analog-to-digital converters.

Objectives

- To generate a program to use A/D converters
- To enter and execute a program for use with A/D converters

Equipment

- MMD-2 Mini-Micro Design/Trainer
- EID-1 Experimental Interface Designer
- MMD-2 to EID-1 Interface Ribbon Cable

Introduction

As you should know, an analog-to-digital converter is a device that converts an analog signal (speed, temperature, light intensity, etc.) into a digital signal. The digital signal is then used by a microprocessor to make a decision.

Located on the EID-1 device are three analog type input devices, a temperature sensing circuit, a photocell for light intensity, and a precision slidepot. Each of these devices is switch selectable and connects that device to the analog-to-digital converter. When the microprocessor (MMD-2) is programmed to do so, the A/D converter is an input device to the microprocessor.

You will use each of the analog inputs and see the effects they have. You will also take some digital measurements of these devices for use in a later lab.

Procedure

Step 1. Connect the EID-1 to the MMD-2.

NOTE: MMD-2 Power MUST be OFF.

Use care when inserting the ribbon cable connectors to avoid damage to the connector pins. Also be sure of the pin sequence alignment (pin 1 to pin 1).

Step 2. Check EID-1 and MMD-2 connections and apply power to the MMD-2.

Step 3. On the EID-1, select the POSITION SENSOR using the select switch S2. Slide the position sensor all the way to the left.

When working with A/D converters, several things must be done in sequence in order to have the A/D input data to the microprocessor. These are:

- A *Start Convert* instruction
- A *Converter Ready* status
- *Input* data

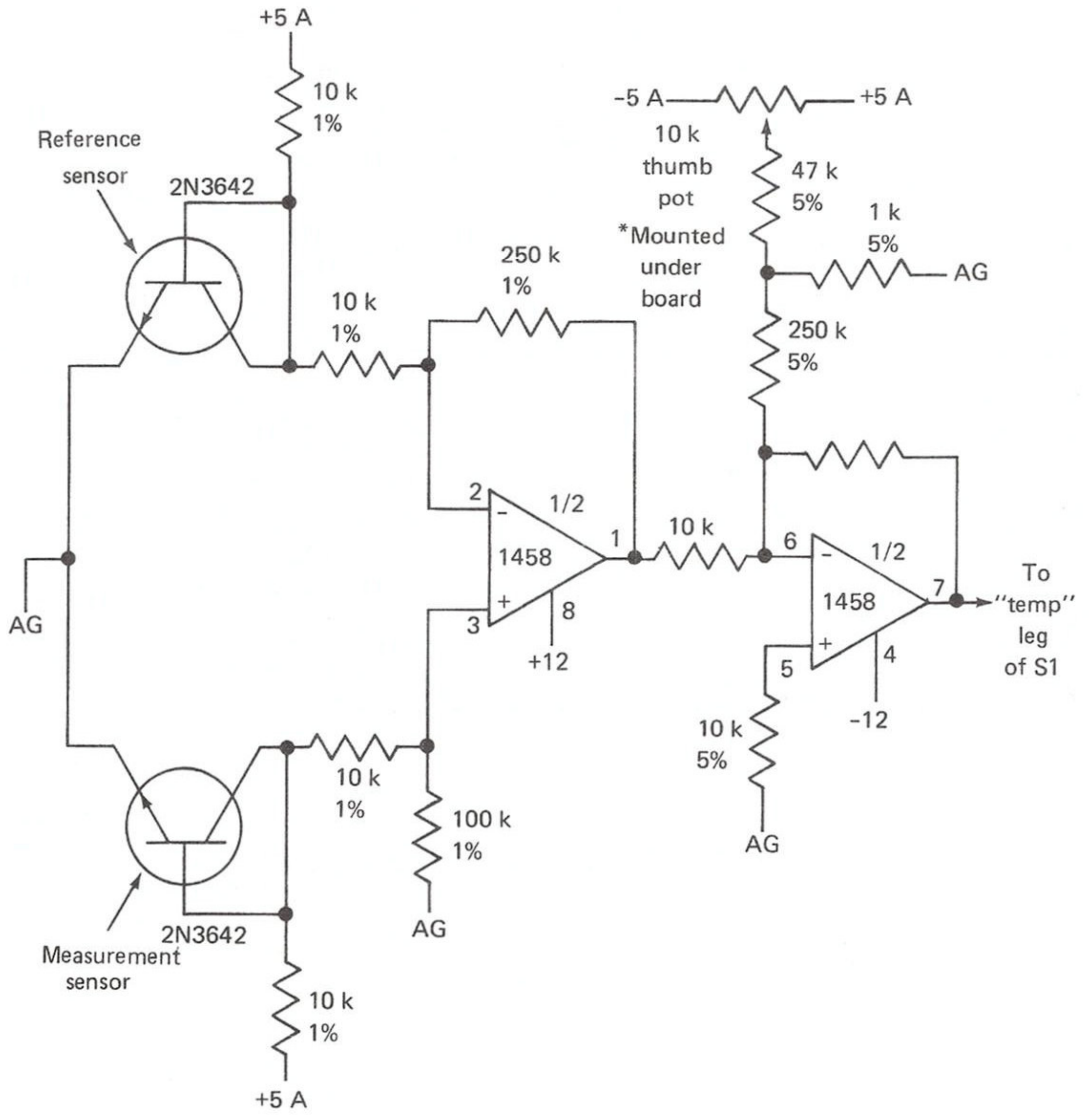
The Start Convert instruction is an output to the device address. The address you will use is **371.** The Converter Ready status is an input instruction from the device (A/D). This address is **371.** The data input is also an input instruction. The address for this instruction is **370.**

The program to input data from this A/D converter is:

```
Code    Mnemonic    Comment

 323    OUT         Start convert.
 371
 333    IN          Input READY status.
 371
 346    ANI         AND ready with 001.
 001
 312    JNZ         If not zero, device
 NNN                is not ready. Go back
 NNN                and check again.
 333    IN          Input data when ready.
 370
```

The following diagrams for the A/D converter show the analog input devices and the A/D converter circuit. (See figures 15 through 17).



Calibration
1) Turn on
2) Warm up 5 min
3) Adj pot to yield +2.0 VDC at "temp" leg of S1
4) Seal pot with glyptol

Figure 15. EID-1 Temperature Input to ADC

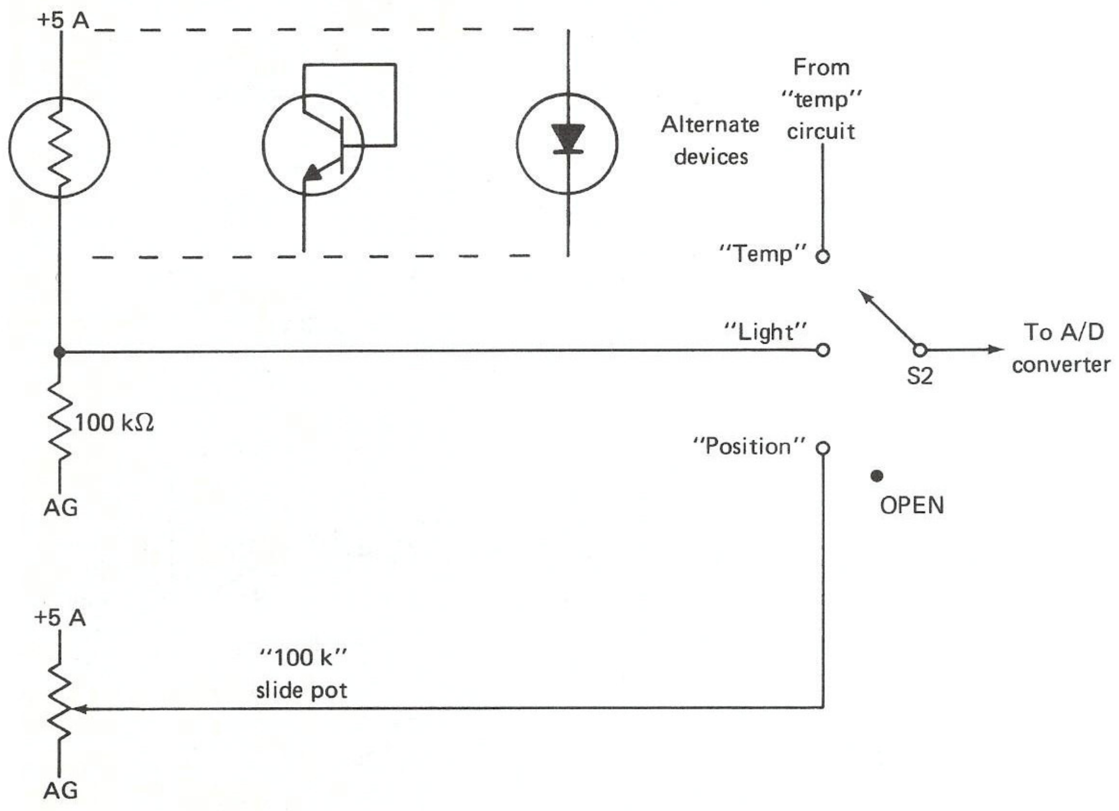


Figure 16. EID-1 Light and Position Input to ADC

Reprinted courtesy of E&L Instruments, Inc.

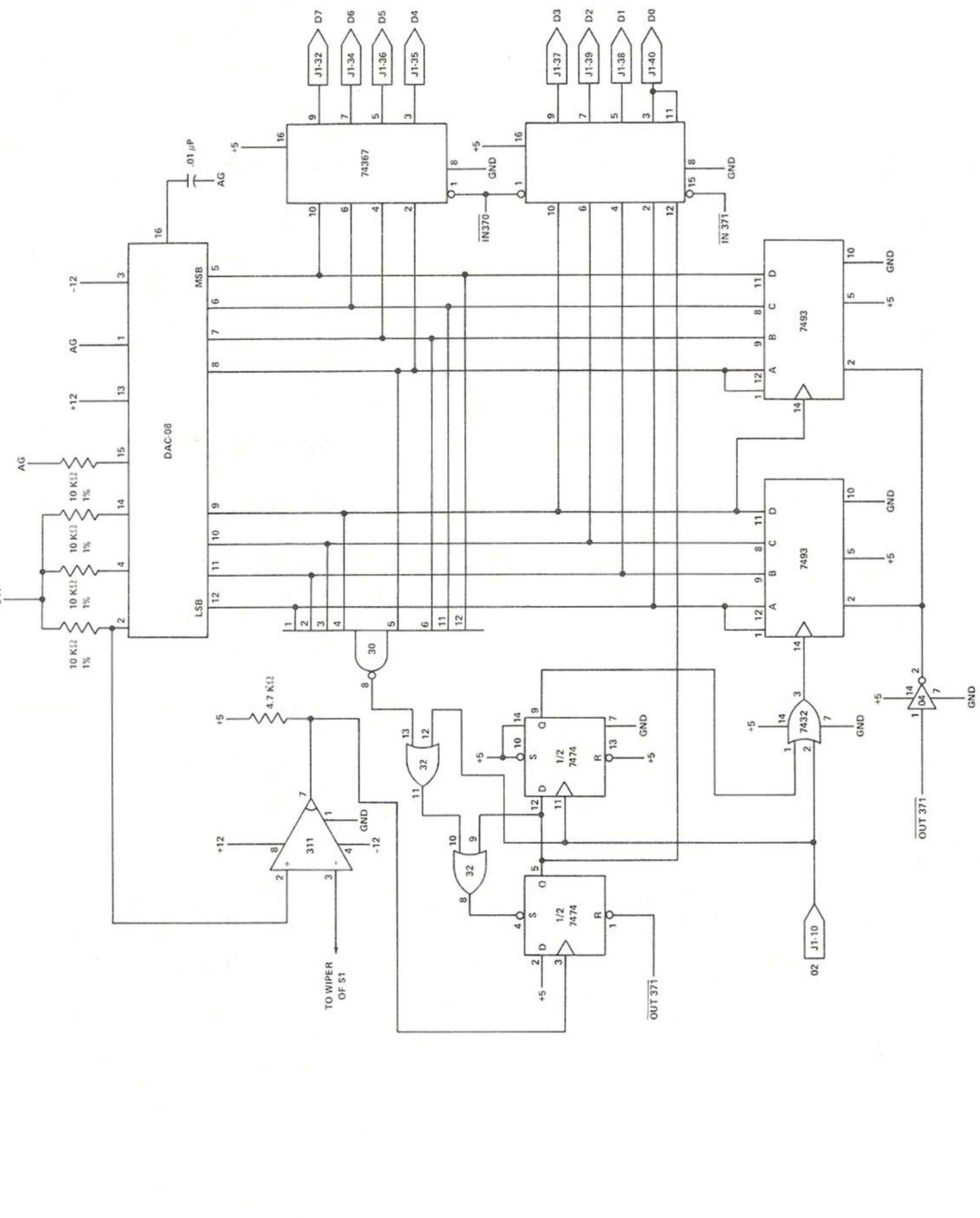


Figure 17. Analog-to-Digital Converter

Reprinted courtesy of E&L Instruments, Inc.

Step 4. Let's write a program to get data from the A/D converter and display it on the EID output LEDs (Port 373).

You could use the program as listed above and add an output to Port 373 and jump back. However, you would find it easier to use a subroutine for the A/D status and data input. This subroutine will remain constant throughout this lab, whereas the other programs will change. Therefore, your program should look something like:

```
Address   Code   Label   Mnemonic          Comment

003000    323            OUT               Start convert.
  3001    371
  3002    315    LOOP    CALL              Get the subroutine
  3003    200                              at 003200.
  3004    003
  3005    323            OUT               Display to LEDs.
  3006    373
  3007    303            JMP LOOP          Go do it over.
  3010    002
  3011    003

                    ANALOG IN SUBROUTINE

003200    333            IN                Ready status in.
  3201    371
  3202    346            ANI               Ready there?
  3203    001
  3204    312            JNZ ANALOGIN      No, go back.
  3205    200                              Yes, go on.
  3206    003
  3207    333            IN                Get data.
  3210    370
  3211    323            OUT               Start convert again.
  3212    371
  3213    311            RET               Go back to main
                                           program
```

Step 5. Enter and execute the program.

Step 6. While the program is running, slide the position sensor slide back and forth a few times while watching the Experimental Interface Designer's LEDs. As you can see, different binary numbers appear in the LEDs as the position is changed.

The A/D converter works exactly like the D/A converter in the previous lab, except that it takes a voltage and turns it into a binary number; the D/A converter turns a binary code into a voltage. Both the D/A and A/D converters on your analog interface board use exactly the same code:

	Bit Position	Voltage
Most Significant	8	5.0
	7	2.5
	6	1.25
	5	0.625
	4	0.3125
	3	0.15625
	2	0.078125
Least Significant	1	0.0390625

When the position indicator is all the way to the left, and all the LEDs are off or zeros, the voltage is zero. When the position sensor is all the way to the right, all the LEDs should be on and the voltage is near 9.9609375 volts or, more approximately, ten volts.

Step 7. Now see if you can set the position sensor to obtain 7.5 volts (bits 8 and 7 on). Can you set the position sensor to any of the other values?____________________(yes)

Remember though, because the A/D converter 'works' in definite steps, there is a slight amount of play in these settings and may be difficult to obtain exact settings.

Step 8. Set the INPUT switch to the PHOTOCELL position. This is a light sensor.

Question. What is the level in the room as displayed on the LEDs?

____

(Remember, this reading will vary depending on the various environmental conditions you are experiencing. For example, fluorescent lamps or incandescent, distance from the source, etc.)

Question. What voltage does this represent? ____________________

Step 9. Cover the PHOTOCELL with your hand.

Question. What is the level as displayed now?____________________

Question. Is the voltage more or less than room light?____________________

Step 10. Set the INPUT switch to the thermistor temperature sensor.

Question. What is the binary temperature as displayed?____________________

Step 11. Put your fingers around the temperature sensor. NOTE: Because there is very little voltage and current present, you will NOT be affected. However, be careful that you do not damage the circuit by moving the temperature sensor around. The component leads could break.

Question. What happens to the binary display as you touch the temperature sensor?

____(increase)

Question. When you let go of the temperature sensor, does the display go back to the old value immediately?

______________ (no)

The light and temperature sensors have some special properties of their own. They do not always fall smoothly in the range of voltages that we might like. Sometimes it is necessary to scale them by adding, subtracting, or multiplying and dividing a constant correction factor. The computer makes this easy for us because of its computational capabilities.

The following program will enable you to see the data (digital) that is output by the A/D converter to the microprocessor.

This program will clear the MMD-2's seven-segment displays, setting them to all zeros. It will then get the A/D signal input and convert it to the octal/hex code, and display that value in the leftmost three displays (HIGH address). To do most of these tasks, the program will use some of the PROM (KEX monitor) program. You will notice the least significant digit of the reading is not very "stable." The stability of this digit is controlled by the time delay routine. The longer the delay, the more stable the information will become. However, the response time will also become longer. This means you will have to wait for the reading to settle before recording.

Step 12. Complete the comment field of the following program; then enter and execute the program.

Address	Code	Mnemonic	Comment
003000	323	**OUT**	
3001	371		
3002	076	**MVI A**	
3003	000		
3004	041	**LXI H**	
3005	000		
3006	374		
3007	167	**MOV M,A**	
3010	054	**INR L**	
3011	167	**MOV M,A**	
3012	054	**INR L**	
3013	167	**MOV M,A**	
3014	315	**CALL** ANALOGIN	
3015	200		
3016	003		
3017	167	**MOV M,A**	
3020	315	**CALL** HLDOUT	
3021	375		
3022	343		
3023	315	**CALL** DELAY	
3024	100		
3025	003		
3026	303	**JMP**	
3027	014		
3030	003		

TIME DELAY ROUTINE

Address	Code	Mnemonic	Comment
003100	117	**MOV C,A**	
3101	021	**LXI D**	
3102	050		
3103	002		
3104	033	**DCX D**	
3105	172	**MOV A,D**	
3106	263	**ORA E**	
3107	302	**JNZ**	
3110	104		
3111	003		
3112	171	**MOV A,C**	
3113	311	**RET**	

The ANALOGIN routine is the one you should have already entered for this lab. However, it is repeated here so you can verify it.

ANALOGIN ROUTINE

```
003200      333     IN
            371
            346     ANI
            001
            312     JNZ
            200
            003
            333     IN
            370
            323     OUT
            371
            311     RET
```

You can now see the various effects each of the analog sensing devices has. This is displayed in the same base code you entered your program in (octal/hex). You will now take some readings, using these input devices. These should be taken as accurately as possible because you will use them in the next lab. As these readings vary from environment to environment, there is no standard or correct reading.

Step 13. Set the INPUT switch to PHOTOCELL. Record the ambient room light (photocell uncovered). ______________________________
Cover the photocell with your hand and record the reading. ______________________________

Step 14. Set the INPUT switch to the temperature sensor. Record the ambient room temperature reading. ______________________________

Step 15. Put your fingers on the temperature sensor. Record the reading after a few minutes. ______________________________

Question. How close to the binary readings you took earlier are these readings?

____

(These readings should be fairly close. However, any differences would be due to length of time you let the device "settle" before the measurement was taken.)

Step 16. Set the INPUT switch to POSITION SENSOR. Vary the slide on this sensor while watching the display.

Step 17. Modify the delay count in the time delay routine at location 003103 to different values. Notice the different stability and response rates.

Summary

In this lab you experimented with three different analog measuring devices. Using an A/D converter, you entered and executed several different programs to see the digital data representing the analog output of these devices.

You learned that to get the A/D converter to work, you had to: output a Start Convert signal; check the Ready Status of the A/D; and if it was ready, input the data. Feel free to experiment on your own using any of the programs developed in this lab.

DAC and ADC Programming Lab

Purpose

The purpose of this lab is to provide you with some experience with programming DAC and ADC control programs.

Objectives

- To write, code, and document control programs
- To enter and execute the programs on the microprocessor trainer

Equipment

- MMD-2 Mini-Micro Design/Trainer
- EID-1 Experimental Interface Designer
- MMD-2 to EID-1 Interface Ribbon Cable

Introduction

As you know and have seen from previous labs, analog signals can be converted into a digital quantity. Similarly, a digital quantity can be converted into an analog value. When these two processes are combined and controlled by a computer or a microprocessor, in a process control application, the A/D is monitoring some process, and if something changes the A/D will inform the microprocessor. The microprocessor, in turn, will make a decision about whether or not the change needs corrective action. If corrective action is needed, the microprocessor will output the change required to the D/A device. The D/A will then make the necessary analog change.

Procedure

Step 1. Write a program that uses the position control sensor to control the speed and output time of the motor. This program should meet the following parameters:

- Start the motor at zero
- Check the position sensor value
- Store this value as a timing byte
- When the time is up, output the value to the motor
- Go back and get the next position sensor value and repeat the process

This program should require nineteen locations, plus the ANALOGIN and DELAY subroutines previously programmed in other labs. Put the ANALOGIN routine at location 003200 and the DELAY at location 003100. The DELAY should be long enough to 'see' the change. Location 3103 should be 100. (Compare your program with that listed in the Answer Key located at the end of this lab.)

Address	Code	Label	Mnemonic	Comments

Step 2. Write a program that will monitor either the light sensor or the temperature sensor. Use the coding sheets provided in this lab (next few pages). The following conditions should be met:

A) With ambient room light/temp, the green lamps (LEDs) on the EID-1, and all Port 2 lamps (LEDs) of the MMD-2 are on. The MOTOR switch is ON. (Remember, if you use the light sensor, it will have maximum output with light and will decrease as it gets darker, and the motor will run at maximum.)

B) With light/temp at a slightly higher setting, the green and Port 2 lamps are off. The yellow lamp and Port 0 lamps are on. The motor starts tracking input (i.e., as temperature continues to increase, motor speed increases.)

C) With light/temp at a second higher setting, yellow and Port 0 lamps are off. The red lamp and Port 1 lamps flash, and the motor continues tracking input.

D) With the light/temp greater than, or equal to the value of, a third-level setting, the red and Port 1 lamps turn off, the motor goes to maximum speed, and the microprocessor halts.

The threshold settings for each of these parameters should be as follows:

- The A parameter is the ambient value that was recorded in the A/D lab for the sensor you are using; *add* a small count to allow for tolerance settings.
- The B parameter is halfway between minimum and maximum values obtained in the A/D lab for the sensor you are using.
- The C parameter is halfway between B parameter setting and maximum value obtained in the A/D lab for the sensor you are using.
- The D parameter is the maximum value minus a small count.

Step 3. Enter and execute your program.

Compare your program with the Answer Key located at the end of this lab. This is one possible solution program.

Address	Code	Label	Mnemonic	Comments

Address	Code	Label	Mnemonic	Comments

Address	Code	Label	Mnemonic	Comments

Address	Code	Label	Mnemonic	Comments

Address	Code	Label	Mnemonic	Comments

Address	Code	Label	Mnemonic	Comments

Step 4. Enter the following program.†

Address	Code	Label	Mnemonic	Comments
003 000	041	LEARN	**LXI H**	Set pointer to
001	000	*		bottom of learn
002	000	*		table.
003	323		**OUT**	Clear A/D Converter.
004	371			
005	333	A/D	**IN**	Do A/D Conversion.
006	371			
007	346		**ANI**	Check Status For
010	001			Conversion Complete.
011	312		**JZ**	
012	005	*		
013	003	*		
014	333		**IN**	Conversion complete,
015	370			fetch movement data.
016	167	STORE	**MOV M, A**	Store in learn
017	315		**CALL**	table.
020	037	*	END	Out of storage?
021	003	*	CHECK	
022	303		**JMP**	No, fetch next
023	003	*		movement.
024	003	*		
025	041	MIMIC	**LXI H**	Set pointer to
026	000	*		bottom of learn
027	000	*		table.
030	176		**MOV A, M**	Fetch movement from
031	315		**CALL**	table; has all
032	037	*	END	movement been
033	003	*	CHECK	mimicked?
034	303		**JMP**	No, do next
035	030	*		movement.
036	003	*		
037	323	END	**OUT**	Send motor speed to
040	370	CHECK		D/A.
041	323		**OUT**	Show binary of
042	373			speed on LEDs.
043	043		**INX H**	Increment table
044	174		**MOV A, H**	pointer. Check for
045	376		**CPI**	end of table.
046	003			
047	302		**JNZ**	Not end, wait a
050	056	*	TIMER	while or we will
051	003	*		run through the demo
052	227		**SUB A**	in just a few
053	323		**OUT**	seconds. Done, turn
054	370			motor off.

†Reprinted courtesy of E & L Instruments, Inc.

Address	Code	Label	Mnemonic	Comments
003 055	166		**HALT**	Show's over.
056	021	TIMER	**LXI D**	Set up a counter
057	000			to chew up time by
060	004			down counting to
061	172		**MOV A, D**	zero.
062	247		**ANA, A**	Test mechanism for register pair.
063	310		**RZ**	Timer done; Return.
064	033		**DCX D**	Timer not done, down count.
065	303		**JMP**	Repeat timer test.
066	061	*		
067	003	*		

Many microcomputers are used to control industrial processes these days. In some applications, microcomputers are set up to be self-teaching; that is, an operator guides some machine through its paces. Meanwhile, the microprocessor is tracking the events and storing them in its memory. When a special program is run by the operator, the microprocessor looks up the stored values and, consequently, makes the machine imitate the procedures the operator did. In one sense, the microprocessor taught itself what to do by following the actions of the machine operator.

Mimic is a special program that lets you demonstrate the self-teaching aspect of your microprocessor. The *mimic* program† is reprinted with permission from the programmer, Matt Veslocki, E & L Instruments, Inc. When you execute this program, you will have about one-half minute to vary the position sensor in any manner you desire. At the end of this delay, the motor and program will stop. You will now have taught the microprocessor what you wanted it to learn. When you initiate the mimic part of the program, it will repeat exactly what you did.

Step 5. Set the INPUT switch to POSITION sensor and MOTOR switch on.

Step 6. Start the program at location 003000.

Step 7. For the next one-half minute, vary the position sensor in any manner or speed you want.

Step 8. When the program stops, set the address to 003025 and run. The microprocessor will play back the exact sequence and speed you entered in step 7.

†Courtesy of E & L Instruments, Inc.

Summary

In this lab you generated two programs. The first program demonstrated how the setting of the position sensor could vary the time delay of the start-up speed on the motor. That is, the output of the A/D converter placed the digital value for the time delay into the microprocessor. The microprocessor then used this value to set the time between output signals to the motor.

The second program you wrote monitored one of the sensors and controlled the output conditions. This program could represent a thermostat control for heating--if the temperature went too high, the system would stop the process, leaving a fan on to cool down. It could also represent a form of "intrusion detector" that sets off an alarm whenever a shadow is cast by an intruder.

There are various applications for this type of program. Finally, you entered a program that tracked and stored all the different inputs. When requested to do so, the microprocessor played back the exact same settings.

This type of program finds widespread application on various assembly lines where numerous repetitive steps are controlled by the processor.

Answer Key

Step 1.

```
Address   Code   Mnemonic         Comment

003000    323    OUT              Start convert.
  3001    371
  3002    006    MVI B            Set ramp to zero.
  3003    000
  3004    170    MOV A,B          Get ready to output.
  3005    323    OUT              Output.
  3006    370
  3007    315    CALL ANALOGIN    Get analog sig.
  3010    200
  3011    003
  3012    062    STA              Store it.
  3013    103
  3014    003
  3015    315    CALL DELAY       Get delay time.
  3016    100
  3017    003
  3020    303    JMP              Go back; do it again.
  3021    005
  3022    003
```

Step 2.

```
Address   Code   Mnemonic         Comment

003000    323    OUT              Start convert.
  3001    371
  3002    315    CALL ANALOGIN    Get analogin routine.
  3003    200
  3004    003
  3005    376    CPI              Compare values.
  3006    NNN (value you supply)  A parameter value.
  3007    332    JC               Jump if less.
  3010    000                     Green light.
  3011    002
  3012    376    CPI              Compare values.
  3013    NNN (value you supply)  B parameter value.
  3014    332    JC               Jump if less.
  3015    050                     Yellow light.
  3016    002
```

Address	Code	Mnemonic	Comments
3017	376	**CPI**	Compare value.
3020	NNN (value you supply)		C parameter value.
3021	332	**JC**	Jump if less.
3022	100		Red light flash.
3023	002		
3024	376	**CPI**	Compare value.
3025	NNN (value you supply)		D parameter value.
3026	332	**JC**	Jump if less.
3027	150		Red light on.
3030	002		
3031	166	**HLT**	Stop.

Step 2. Subroutines

Green Light

Address	Code	Mnemonic	Comments
002000	076	**MVI A**	Green light data.
2001	007		
2002	323	**OUT**	Green lights out.
2003	373		
2004	076	**MVI A**	Port 2 data.
2005	377		
2006	323	**OUT**	Port 2 out.
2007	002		
2010	076	**MVI A**	Port 2 data.
2011	000		
2012	323	**OUT**	Port 2 out.
2013	002		
2014	303	**JMP**	Go back; do it again.
2015	002		
2016	003		

Yellow Light

Address	Code	Mnemonic	Comments
002050	323	**OUT**	Output to the motor.
2051	370		
2052	076	**MVI A**	Yellow light data.
2053	070		
2054	323	**OUT**	Yellow lights out.
2055	373		
2056	076	**MVI A**	Port 0 data.
2057	377		
2060	323	**OUT**	Port 0 out.
2061	000		
2062	076	**MVI A**	Port 0 data.
2063	000		
2064	323	**OUT**	Port 0 out.
2065	000		
2066	303	**JMP**	Go back; do it again.
2067	002		
2070	003		

Red Light (flashing)

```
Address   Code   Mnemonic   Comments

002100    323    OUT        Output to the motor.
  2101    370
  2102    076    MVI A      Red light data.
  2103    300
  2104    323    OUT        Red light out.
  2105    373
  2106    076    MVI A      Port 1 data.
  2107    377
  2110    323    OUT        Port 1 out.
  2111    001
  2112    315    CALL       Get ON time.
  2113    050
  2114    001
  2115    076    MVI A      Red light data.
  2116    000
  2117    323    OUT        Red light out.
  2120    373
  2121    323    OUT        Port 1 out.
  2122    001
  2123    315    CALL       Get OFF time.
  2124    050
  2125    001
  2126    303    JMP        Go back; do it again.
  2127    002
  2130    003
```

Red Light On

```
Address   Code   Mnemonic   Comments

002150    323    OUT        Motor out.
  2151    370
  2152    076    MVI A      Red light data.
  2153    300
  2154    323    OUT        Red light out.
  2155    373
  2156    076    MVI A      Port 1 data.
  2157    377
  2160    323    OUT        Port 1 out.
  2161    001
  2162    303    JMP        Go back; do it again.
  2163    002
  2164    003
```

ANALOGIN

```
003200   333   IN          Ready status.
  3201   371
  3202   346   ANI         Ready match.
  3203   001
  3204   312   JNZ         No, go back.
  3205   200
  3206   003
  3207   333   IN          Data in.
  3210   370
  3211   323   OUT         Start convert again.
  3212   371
  3213   311   RET         Go back.
```

Red Light Timer

```
001050   117   MOV C, A    Save the data in A.
  1051   021   LXI D       Set up time delay,
  1052   050
  1053   020
  1054   033   DCX D       Count down one.
  1055   172   MOV A, D    Move D to A.
  1056   263   ORA E       OR it with E.
  1057   302   JNZ         Not zero; go back.
  1060   054
  1061   001
  1062   171   MOV A, C    Get the data saved.
  1063   311   RET         Return to the program.
```

(Blank Page)

Appendix

(Blank Page)

Appendix A
8080 Instruction Set*

A computer, no matter how sophisticated, can only do what it is "told" to do. One "tells" the computer what to do via a series of coded instructions referred to as a **Program.** The realm of the programmer is referred to as **Software,** in contrast to the **Hardware** that comprises the actual computer equipment. A computer's software refers to all of the programs that have been written for that computer.

When a computer is designed, the engineers provide the Central Processing Unit (CPU) with the ability to perform a particular set of operations. The CPU is designed such that a specific operation is performed when the CPU control logic decodes a particular instruction. Consequently, the operations that can be performed by a CPU define the computer's **Instruction Set.**

Each computer instruction allows the programmer to initiate the performance of a specific operation. All computers implement certain arithmetic operations in their instruction set, such as an instruction to add the contents of two registers. Often logic operations (e.g., OR the contents of two registers) and register operate instructions (e.g., increment a register) are included in the instruction set. A computer's instruction set will also have instructions that move data between registers, between a register and memory, and between a register and an I/O device. Most instruction sets also provide **Conditional Instructions.** A conditional instruction specifies an operation to be performed only if certain conditions have been met; for example, jump to a particular instruction if the result of the last operation was zero. Conditional instructions provide a program with a decision-making capability.

By logically organizing a sequence of instructions into a coherent program, the programmer can "tell" the computer to perform a very specific and useful function.

The computer, however, can only execute programs whose instructions are in a binary coded form (i.e., a series of 1's and 0's), that is called **Machine Code.** Because it would be extremely cumbersome to program in machine code, programming languages have been developed. There are programs available which convert the programming language instructions into machine code that can be interpreted by the processor.

One type of programming language is **Assembly Language.** A unique assembly language mnemonic is assigned to each of the computer's instructions. The programmer can write a program (called the **Source Program**) using these mnemonics and certain operands; the source program is then converted into machine instructions (called the **Object Code**). Each assembly language instruction is converted into one machine code instruction (1 or more bytes) by an **Assembler** program. Assembly languages are usually machine dependent (i.e., they are usually able to run on only one type of computer).

*Reprinted by permission of Intel Corporation, copyright 1977 and 1975

The 8080 Instruction Set

The 8080 instruction set includes five different types of instructions:

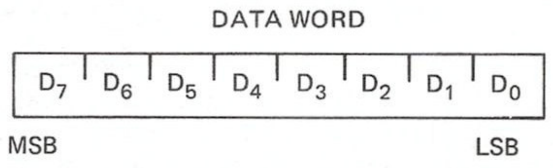
- **Data Transfer Group**—move data between registers or between memory and registers
- **Arithmetic Group**—add, subtract, increment or decrement data in registers or in memory
- **Logical Group**—AND, OR, EXCLUSIVE-OR, compare, rotate or complement data in registers or in memory
- **Branch Group**—conditional and unconditional jump instructions, subroutine call instructions and return instructions
- **Stack, I/O and Machine Control Group**—includes I/O instructions, as well as instructions for maintaining the stack and internal control flags.

Instruction and Data Formats:

Memory for the 8080 is organized into 8-bit quantities, called Bytes. Each byte has a unique 16-bit binary address corresponding to its sequential position in memory.

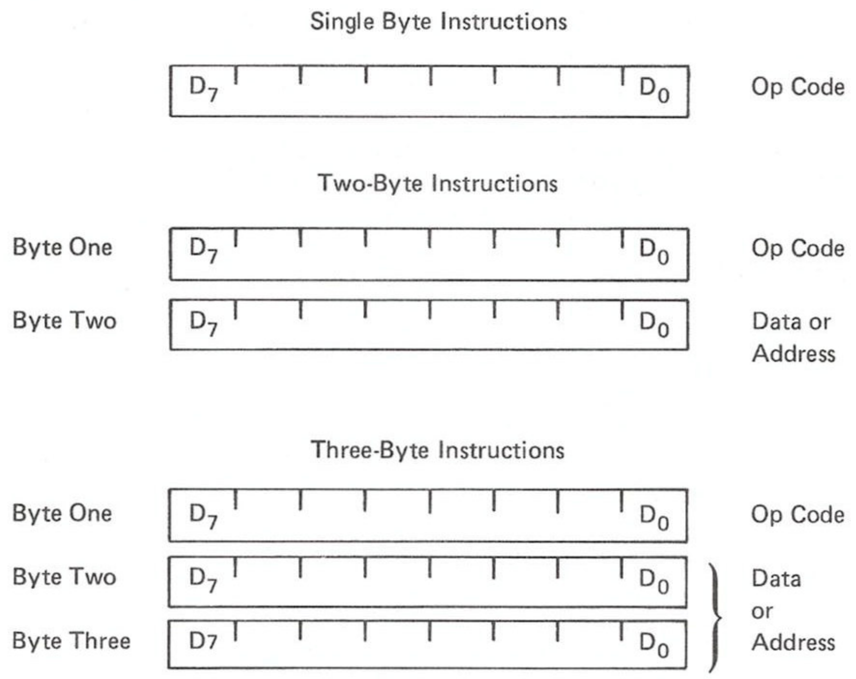
The 8080 can directly address up to 65,536 bytes of memory, which may consist of both read-only memory (ROM) elements and random-access memory (RAM) elements (read/write memory).

Data in the 8080 is stored in the form of 8-bit binary integers:



When a register or data word contains a binary number, it is necessary to establish the order in which the bits of the number are written. In the Intel 8080, BIT 0 is referred to as the **Least Significant Bit (LSB),** and BIT 7 (of an 8 bit number) is referred to as the **Most Significant Bit (MSB).**

The 8080 program instructions may be one, two or three bytes in length. Multiple byte instructions must be stored in successive memory locations; the address of the first byte is always used as the address of the instructions. The exact instruction format will depend on the particular operation to be executed.



Addressing Modes:

Often the data that is to be operated on is stored in memory. When multi-byte numeric data is used, the data, like instructions, is stored in successive memory locations, with the least significant byte first, followed by increasingly significant bytes. The 8080 has four different modes for addressing data stored in memory or in registers:

- Direct—Bytes 2 and 3 of the instruction contain the exact memory address of the data item (the low-order bits of the address are in byte 2, the high-order bits in byte 3).
- Register—The instruction specifies the register or register-pair in which the data is located.
- Register Indirect—The instruction specifies a register-pair which contains the memory address where the data is located (the high-order bits of the address are in the first register of the pair, the low-order bits in the second).
- Immediate—The instruction contains the data itself. This is either an 8-bit quantity or a 16-bit quantity (least significant byte first, most significant byte second).

Unless directed by an interrupt or branch instruction, the execution of instructions proceeds through consecutively increasing memory locations. A branch instruction can specify the address of the next instruction to be executed in one of two ways:

- Direct—The branch instruction contains the address of the next instruction to be executed. (Except for the 'RST' instruction, byte 2 contains the low-order address and byte 3 the high-order address.)
- Register indirect—The branch instruction indicates a register-pair which contains the address of the next instruction to be executed. (The high-order bits of the address are in the first register of the pair, the low-order bits in the second.)

The RST instruction is a special one-byte call instruction (usually used during interrupt sequences). RST includes a three-bit field; program control is transferred to the instruction whose address is eight times the contents of this three-bit field.

Condition Flags:

There are five condition flags associated with the execution of instructions on the 8080. They are Zero, Sign, Parity, Carry, and Auxiliary Carry, and are each represented by a 1-bit register in the CPU. A flag is "set" by forcing the bit to 1; "reset" by forcing the bit to 0.

Unless indicated otherwise, when an instruction affects a flag, it affects it in the following manner:

Zero:	If the result of an instruction has the value 0, this flag is set; otherwise it is reset.
Sign:	If the most significant bit of the result of the operation has the value 1, this flag is set; otherwise it is reset.
Parity:	If the modulo 2 sum of the bits of the result of the operation is 0, (i.e., if the result has even parity), this flag is set; otherwise it is reset (i.e., if the result has odd parity).
Carry:	If the instruction resulted in a carry (from addition), or a borrow (from subtraction or a comparison) out of the high-order bit, this flag is set; otherwise it is reset.
Auxiliary Carry:	If the instruction caused a carry out of bit 3 and into bit 4 of the resulting value, the auxiliary carry is set; otherwise it is reset. This flag is affected by single precision additions, subtractions, increments, decrements, comparisons, and logical operations, but is principally used with additions and increments preceding a DAA (Decimal Adjust Accumulator) instruction.

Symbols and Abbreviations:

The following symbols and abbreviations are used in the subsequent description of the 8080 instructions:

SYMBOLS	MEANING
accumulator	Register A
addr	16-bit address quantity
data	8-bit data quantity
data 16	16-bit data quantity
byte 2	The second byte of the instruction
byte 3	The third byte of the instruction
port	8-bit address of an I/O device
r,rl,r2	One of the registers A,B,C,D,E,H,L
DDD,SSS	The bit pattern designating one of the registers A,B,C,D,E,H,L (DDD=destination,SSS=source):

DDD or SSS	REGISTER NAME
111	A
000	B
001	C
010	D
011	E
100	H
101	L

rp	One of the register pairs: B represents the B,C pair with B as the high-order register and C as the low-order register; D represents the D,E pair with D as the high-order register and E as the low-order register; H represents the H,L pair with H as the high-order register and L as the low-order register; SP represents the 16-bit stack pointer register.
RP	The bit pattern designating one of the register pairs B,D,H,SP;

RP	REGISTER PAIR
00	B-C
01	D-E
10	H-L
11	SP

rh	The first (high-order) register of a designated register pair.
rl	The second (low-order) register of a designated register pair.

SYMBOLS	MEANING
PC	16-bit program counter register (PCH and PCL are used to refer to the high-order and low-order 8 bits respectively).
SP	16-bit stack pointer register (SPH and SPL are used to refer to the high-order and low-order 8 bits respectively).
r_m	Bit m of the register r (bits are number 7 through 0 from the left to right).
Z,S,P,CY,AC	The condition flags: Zero, Sign, Parity, Carry, and Auxiliary Carry, respectively.
( )	The contents of the memory location or registers are enclosed in the parentheses.
$\leftarrow$	"Is transferred to"
$\wedge$	Logical AND
$\forall$	Exclusive OR
$\vee$	Inclusive OR
+	Addition
–	Two's complement subtraction
*	Multiplication
$\longleftrightarrow$	"Is exchanged with"
$\overline{}$	The one's complement (e.g., $(\overline{A})$)
n	The restart number 0 through 7
NNN	The binary representation 000 through 111 for restart number 0 through 7 respectively.

Description Format

The following pages provide a detailed description of the instruction set of the 8080. Each instruction is described in the following manner:

1. The MAC 80 assembler format, consisting of the instruction mnemonic and operand fields, is printed in **BOLDFACE** on the left side of the first line.
2. The name of the instruction is enclosed in parenthesis on the right side of the first line.

3. The next line(s) contain a symbolic description of the operation of the instruction.
4. This is followed by a narrative description of the operation of the instruction.
5. The following line(s) contain the binary fields and patterns that comprise the machine instruction.
6. The last four lines contain incidental information about the execution of the instruction. The number of machine cycles and states required to execute the instruction are listed first. If the instruction has two possible execution times, as in a Conditional Jump, both times will be listed, separated by a slash. Next, any significant data addressing modes (see pp. A3 and A4) are listed. The last line lists any of the five Flags that are affected by the execution of the instruction.

Data Transfer Group

This group of instructions transfers data to and from registers and memory. Condition flags are not affected by any instruction in this group.

MOV r1, r2 (Move Register)

(r1) ← (r2)

The content of register r2 is moved to register r1.

0	1	D	D	D	S	S	S

Cycles:	1
States:	5
Addressing:	register
Flags:	none

MOV r, M (Move from memory)

(r) ← ((H) (L))

The content of the memory location, whose address is in registers H and L, is moved to register r.

0	1	D	D	D	1	1	0

Cycles:	2
States:	7
Addressing:	reg. indirect
Flags:	none

MOV M, r (Move to memory)

((H) (L)) ← (r)

The content of register r is moved to the memory location whose address is in registers H and L.

0	1	1	1	0	S	S	S

Cycles: 2
States: 7
Addressing: reg. indirect
Flags: none

MVI r, data (Move Immediate)

(r) ← (byte 2)

The content of byte 2 of the instruction is moved to register r.

0	0	D	D	D	1	1	0
data							

Cycles: 2
States: 7
Addressing: immediate
Flags: none

MVI M, data (Move to memory immediate)

((H) (L)) ← (byte 2)

The content of byte 2 of the instruction is moved to the memory location whose address is in registers H and L.

0	0	1	1	0	1	1	0
data							

Cycles: 3
States: 10
Addressing: immed./reg. indirect
Flags: none

LXI rp, data 16 (Load register pair immediate)

(rh) ← (byte 3),

(rl) ← (byte 2)

Byte 3 of the instruction is moved into the high-order register (rh) of the register pair rp. Byte 2 of the instruction is moved into the low-order register (rl) of the register pair rp.

0	0	R	P	0	0	0	1
low-order data							
high-order data							

Cycles:	3
States:	10
Addressing:	immediate
Flags:	none

LDA addr (Load Accumulator direct)

(A) ← ((byte 3)(byte 2))

The content of the memory location, whose address is specified in byte 2 and byte 3 of the instruction, is moved to register A.

0	0	1	1	1	0	1	0
low-order addr							
high-order addr							

Cycles:	4
States:	13
Addressing:	direct
Flags:	none

STA addr (Store Accumulator direct)

((byte 3)(byte 2)) ← (A)

The content of the accumulator is moved to the memory location whose address is specified in byte 2 and byte 3 of the instruction.

0	0	1	1	0	0	1	0
low-order addr							
high-order addr							

Cycles: 4
States: 13
Addressing: direct
Flags: none

LHLD addr (Load H and L direct)

(L) ← ((byte 3)(byte 2))

(H) ← ((byte 3)(byte 2) + 1)

The content of the memory location, whose address is specified in byte 2 and byte 3 of the instruction, is moved to register L. The content of the memory location at the succeeding address is moved to register H.

0	0	1	0	1	0	1	0
low-order addr							
high-order addr							

Cycles: 5
States: 16
Addressing: direct
Flags: none

SHLD addr (Store H and L direct)

((byte 3)(byte 2)) ← (L)

((byte 3)(byte 2) + 1) ← (H)

The content of register L is moved to the memory location whose address is specified in byte 2 and byte 3. The content of register H is moved to the succeeding memory location.

0	0	1	0	0	0	1	0
low-order addr							
high-order addr							

Cycles:	5
States:	16
Addressing:	direct
Flags:	none

LDAX rp (Load accumulator indirect)

(A) ← ((rp))

The content of the memory location, whose address is the register pair rp, is moved to register A. Note: only register pairs rp=B (registers B and C) or rp=D (registers D and E) may be specified.

0	0	R	P	1	0	1	0

Cycles:	2
States:	7
Addressing:	reg. indirect
Flags:	none

STAX rp (Store accumulator indirect)

((rp)) ← (A)

The content of register A is moved to the memory location whose address is in the register pair rp. Note: only register pairs rp=B (registers B and C) or rp=D (registers D and E) may be specified.

0	0	R	P	0	0	1	0

Cycles:	2
States:	7
Addressing:	reg. indirect
Flags:	none

XCHG (Exchange H and L with D and E)

(H) ⟷ (D)
(L) ⟷ (E)

The contents of registers H and L are exchanged with the contents of registers D and E.

1	1	1	0	1	0	1	1

Cycles:	1
States:	4
Addressing:	register
Flags:	none

Arithmetic Group:

This group of instructions performs arithmetic operations on data in registers and memory.

Unless indicated otherwise, all instructions in this group affect the Zero, Sign, Parity, Carry, and Auxiliary Carry flags according to the standard rules.

All subtraction operations are performed via two's complement arithmetic and set the carry flag to one to indicate a borrow and clear it to indicate no borrow.

ADD r (Add Register)

$(A) \leftarrow (A) + (r)$

The content of register r is added to the content of the accumulator. The result is placed in the accumulator.

1	0	0	0	0	S	S	S

Cycles:	1
States:	4
Addressing:	register
Flags:	Z,S,P,CY,AC

ADD M (Add memory)

(A) ← (A) + ((H) (L))

The content of the memory location whose address is contained in the H and L registers is added to the content of the accumulator. The result is placed in the accumulator.

1	0	0	0	0	1	1	0

Cycles: 2
States: 7
Addressing: reg. indirect
Flags: Z,S,P,CY,AC

ADI data (Add immediate)

(A) ← (A) + (byte 2)

The content of the second byte of the instruction is added to the content of the accumulator. The result is placed in the accumulator.

1	1	0	0	0	1	1	0
data							

Cycles: 2
States: 7
Addressing: immediate
Flags: Z,S,P,CY,AC

ADC r (Add Register with carry)

(A) ← (A) + (r) + (CY)

The content of register r and the content of the carry bit are added to the content of the accumulator. The result is placed in the accumulator.

1	0	0	0	1	S	S	S

Cycles: 1
States: 4
Addressing: register
Flags: Z,S,P,CY,AC

ADC M (Add memory with carry)

$(A) \leftarrow (A) + ((H)\ (L)) + (CY)$

The content of the memory location whose address is contained in the H and L registers and the content of the CY flag are added to the accumulator. The result is placed in the accumulator.

1	0	0	0	1	1	1	0

Cycles:	2
States:	7
Addressing:	reg. indirect
Flags:	Z,S,P,CY,AC

ACI data (Add immediate with carry)

$(A) \leftarrow (A) + (\text{byte } 2) + (CY)$

The content of the second byte of the instruction and the content of the CY flag are added to the contents of the accumulator. The result is placed in the accumulator.

1	1	0	0	1	1	1	0
data							

Cycles:	2
States:	7
Addressing:	immediate
Flags:	Z,S,P,CY,AC

SUB r (Subtract Register)

$(A) \leftarrow (A) - (r)$

The content of register r is subtracted from the content of the accumulator. The result is placed in the accumulator.

1	0	0	1	0	S	S	S

Cycles:	1
States:	4
Addressing:	register
Flags:	Z,S,P,CY,AC

SUB M (Subtract memory)

(A) ← (A) – ((H) (L))

The content of the memory location whose address is contained in the H and L registers is subtracted from the content of the accumulator. The result is placed in the accumulator.

1	0	0	1	0	1	1	0

Cycles:	2
States:	7
Addressing:	reg. indirect
Flags:	Z,S,P,CY,AC

SUI data (Subtract immediate)

(A) ← (A) – (byte 2)

The content of the second byte of the instruction is subtracted from the content of the accumulator. The result is placed in the accumulator.

1	1	0	1	0	1	1	0
data							

Cycles:	2
States:	7
Addressing:	immediate
Flags:	Z,S,P,CY,AC

SBB r (Subtract Register with borrow)

(A) ← (A) – (r) – (CY)

The content of register r and the content of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.

1	0	0	1	1	S	S	S

Cycles:	1
States:	4
Addressing:	register
Flags:	Z,S,P,CY,AC

SBB M (Subtract memory with borrow)

(A) ← (A) – ((H) (L)) – (CY)

The content of the memory location whose address is contained in the H and L registers and the content of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.

1	0	0	1	1	1	1	0

Cycles: 2
States: 7
Addressing: reg. indirect
Flags: Z,S,P,CY,AC

SBI data (Subtract immediate with borrow)

(A) ← (A) – (byte 2) – (CY)

The contents of the second byte of the instruction and the contents of the CY flag are both subtracted from the accumulator. The result is placed in the accumulator.

1	1	0	1	1	1	1	0
data							

Cycles: 2
States: 7
Addressing: immediate
Flags: Z,S,P,CY,AC

INR r (Increment register)

(r) ← (r) + 1

The content of register r is incremented by one. Note: All condition flags **except CY** are affected.

0	0	D	D	D	1	0	0

Cycles: 1
States: 5
Addressing: register
Flags: Z,S,P,AC

INR M (Increment memory)

((H) (L)) ← ((H) (L)) + 1

The content of the memory location whose address is contained in the H and L registers is incremented by one. Note: All condition flags **except CY** are affected.

0	0	1	1	0	1	0	0

Cycles:	3
States:	10
Addressing:	reg. indirect
Flags:	Z,S,P,AC

DCR r (Decrement Register)

(r) ← (r) – 1

The content of register r is decremented by one. Note: All condition flags **except CY** are affected.

0	0	D	D	D	1	0	1

Cycles:	1
States:	5
Addressing:	register
Flags:	Z,S,P,AC

DCR M (Decrement memory)

((H) (L)) ← ((H) (L)) – 1

The content of the memory location whose address is contained in the H and L registers is decremented by one. Note: All condition flags **except CY** are affected.

0	0	1	1	0	1	0	1

Cycles:	3
States:	10
Addressing:	reg. indirect
Flags:	Z,S,P,AC

INX rp (Increment register pair)

(rh) (rl) ← (rh) (rl) + 1

The content of the register pair rp is incremented by one. Note: **No condition flags are affected.**

0	0	R	P	0	0	1	1

Cycles:	1
States:	5
Addressing:	register
Flags:	none

DCX rp (Decrement register pair)

(rh) (rl) ← (rh) (rl) – 1

The content of the register pair rp is decremented by one. Note: **No condition flags are affected.**

0	0	R	P	1	0	1	1

Cycles:	1
States:	5
Addressing:	register
Flags:	none

DAD rp (Add register pair to H and L)

(H) (L) ← (H) (L) + (rh) (rl)

The content of the register pair rp is added to the content of the register pair H and L. The result is placed in the register pair H and L. Note: **Only the CY flag is affected.** It is set if there is a carry out of the double precision add; otherwise it is reset.

0	0	R	P	1	0	0	1

Cycles:	3
States:	10
Addressing:	register
Flags:	CY

DAA (Decimal Adjust Accumulator)

The eight-bit number in the accumulator is adjusted to form two four-bit Binary-Coded-Decimal digits by the following process:

1. If the value of the least significant 4 bits of the accumulator is greater than 9 or if the AC flag is set, 6 is added to the accumulator.
2. If the value of the most significant 4 bits of the accumulator is now greater than 9, or if the CY flag is set, 6 is added to the most significant 4 bits of the accumulator.

NOTE: All flags are affected.

0	0	1	0	0	1	1	1

Cycles:	1
States:	4
Flags:	Z,S,P,CY,AC

Logical Group:

This group of instructions performs logical (Boolean) operations on data in registers and memory and on condition flags.

Unless indicated otherwise, all instructions in this group affect the Zero, Sign, Parity, Auxiliary Carry, and Carry flags according to the standard rules.

ANA r (AND Register)

(A) ← (A) ∧ (r)

The content of the register r is logically anded with the content of the accumulator. The result is placed in the accumulator. **The CY flag is cleared.**

1	0	1	0	0	S	S	S

Cycles: 1
States: 4
Addressing: register
Flags: Z,S,P,CY,AC

ANA M (AND memory)

(A) ← (A) ∧ ((H) (L))

The contents of the memory location whose address is contained in the H and L registers is logically anded with the content of the accumulator. The result is placed in the accumulator. **The CY flag is cleared.**

1	0	1	0	0	1	1	0

Cycles: 2
States: 7
Addressing: reg. indirect
Flags: Z,S,P,CY,AC

ANI data (AND immediate)

(A) ← (A) ∧ (byte 2)

The content of the second byte of the instruction is logically anded with the contents of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**

1	1	1	0	0	1	1	0
data							

Cycles: 2
States: 7
Addressing: immediate
Flags: Z,S,P,CY,AC

XRA r (Exclusive OR Register)

$(A) \leftarrow (A) \forall (r)$

The content of the register r is exclusive-or'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**

1	0	1	0	1	S	S	S

Cycles: 1
States: 4
Addressing: register
Flags: Z,S,P,CY,AC

XRA M (Exclusive OR Memory)

$(A) \leftarrow (A) \forall ((H)\,(L))$

The content of the memory location whose address is contained in the H and L registers is exclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**

1	0	1	0	1	1	1	0

Cycles: 2
States: 7
Addressing: reg. indirect
Flags: Z,S,P,CY,AC

XRI data (Exclusive OR immediate)

$(A) \leftarrow (A) \forall (\text{byte } 2)$

The content of the second byte of the instruction is exclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**

1	1	1	0	1	1	1	0
data							

Cycles: 2
States: 7
Addressing: immediate
Flags: Z,S,P,CY,AC

ORA r (OR Register)

(A) ← (A) V (r)

The content of register r is inclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**

1	0	1	1	0	S	S	S

Cycles:	1
States:	4
Addressing:	register
Flags:	Z,S,P,CY,AC

ORA M (OR memory)

(A) ← (A) V ((H) (L))

The content of the memory location whose address is contained in the H and L registers is inclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**

1	0	1	1	0	1	1	0

Cycles:	2
States:	7
Addressing:	reg. indirect
Flags:	Z,S,P,CY,AC

ORI data (OR immediate)

(A) ← (A) V (byte 2)

The content of the second byte of the instruction is inclusive-OR'd with the content of the accumulator. The result is placed in the accumulator. **The CY and AC flags are cleared.**

1	1	1	1	0	1	1	0
data							

Cycles:	2
States:	7
Addressing:	immediate
Flags:	Z,S,P,CY,AC

CMP r (Compare Register)

(A) – (r)

The content of register r is subtracted from the accumulator. The accumulator remains unchanged. The condition flags are set as a result of the subtraction. **The Z flag is set to 1 if (A) = (r). The CY flag is set to 1 if (A) < (r).**

1	0	1	1	1	S	S	S

Cycles:	1
States:	4
Addressing:	register
Flags:	Z,S,P,CY,AC

CMP M (Compare memory)

(A) – ((H) (L))

The content of the memory location whose address is contained in the H and L registers is subtracted from the accumulator. The accumulator remains unchanged. The condition flags are set as a result of the subtraction. The Z flag is set to 1 if (A) = ((H) (L)). The CY flag is set to 1 if (A) < ((H) (L)).

1	0	1	1	1	1	1	0

Cycles:	2
States:	7
Addressing:	reg. indirect
Flags:	Z,S,P,CY,AC

CPI data (Compare immediate)

(A) – (byte 2)

The content of the second byte of the instruction is subtracted from the accumulator. The condition flags are set by the result of the subtraction. The Z flag is set to 1 if (A) = (byte 2). The CY flag is set to 1 if (A) < (byte 2).

1	1	1	1	1	1	1	0
data							

Cycles:	2
States:	7
Addressing:	immediate
Flags:	Z,S,P,CY,AC

RLC (Rotate left)

$(A_{n+1}) \leftarrow (A_n)$; $(A_0) \leftarrow (A_7)$
$(CY) \leftarrow (A_7)$

The content of the accumulator is rotated left one position. The low order bit and the CY flag are both set to the value shifted out of the high order bit position. **Only the CY flag is affected.**

0	0	0	0	0	1	1	1

Cycles: 1
States: 4
Flags: CY

RRC (Rotate right)

$(A_n) \leftarrow (A_{n+1})$; $(A_7) \leftarrow (A_0)$
$(CY) \leftarrow (A_0)$

The content of the accumulator is rotated right one position. The high order bit and the CY flag are both set to the value shifted out of the low order bit position. **Only the CY flag is affected.**

0	0	0	0	1	1	1	1

Cycles: 1
States: 4
Flags: CY

RAL (Rotate left through carry)

$(A_{n+1}) \leftarrow (A_n)$; $(CY) \leftarrow (A_7)$
$(A_0) \leftarrow (CY)$

The content of the accumulator is rotated left one position through the CY flag. The low order bit is set equal to the CY flag and the CY flag is set to the value shifted out of the high order bit. **Only the CY flag is affected.**

0	0	0	1	0	1	1	1

Cycles: 1
States: 4
Flags: CY

RAR (Rotate right through carry)

$(A_n) \leftarrow (A_{n+1})$; $(CY) \leftarrow (A_0)$

$(A_7) \leftarrow (CY)$

The content of the accumulator is rotated right one position through the CY flag. The high order bit is set to the CY flag and the CY flag is set to the value shifted out of the low order bit. **Only the CY flag is affected.**

0	0	0	1	1	1	1	1

Cycles: 1
States: 4
Flags: CY

CMA (Complement accumulator)

$(A) \leftarrow (\overline{A})$

The contents of the accumulator are complemented (zero bits become 1, one bits become 0). **No flags are affected.**

0	0	1	0	1	1	1	1

Cycles: 1
States: 4
Flags: none

CMC (Complement carry)

$(CY) \leftarrow (\overline{CY})$

The CY flag is complemented. **No other flags are affected.**

0	0	1	1	1	1	1	1

Cycles: 1
States: 4
Flags: CY

STC (Set carry)

$(CY) \leftarrow 1$

The CY flag is set to 1. **No other flags are affected.**

0	0	1	1	0	1	1	1

Cycles: 1
States: 4
Flags: CY

Branch Groups

This group of instructions alter normal sequential program flow.

Condition flags are not affected by any instruction in this group.

The two types of branch instructions are unconditional and conditional. Unconditional transfers simply perform the specified operation on register PC (the program counter). Conditional transfers examine the status of one of the four processor flags to determine if the specified branch is to be executed. The conditions that may be specified are as follows:

CONDITION	CCC
NZ — not zero (Z = 0)	000
Z — zero (Z = 1)	001
NC — no carry (CY = 0)	010
C — carry (CY = 1)	011
PO — parity odd (P = 0)	100
PE — parity even (P = 1)	101
P — plus (S = 0)	110
M — minus (S = 1)	111

JMP addr (Jump)

(PC) ← (byte 3) (byte 2)

Control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction.

1	1	0	0	0	0	1	1
low-order addr							
high-order addr							

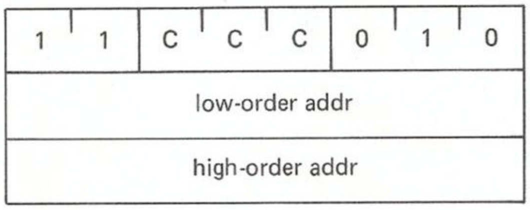
Cycles:	3
States:	10
Addressing:	immediate
Flags:	none

Jcondition addr (Conditional jump)

If (CCC),

(PC) ← (byte 3) (byte 2)

If the specified condition is true, control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction; otherwise, control continues sequentially.

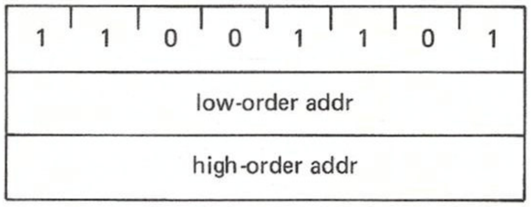


Cycles: 3
States: 10
Addressing: immediate
Flags: none

CALL addr (Call)

((SP) – 1) ← (PCH)
((SP) – 2) ← (PCL)
(SP) ← (SP) – 2
(PC) ← (byte 3) (byte 2)

The high-order eight bits of the next instruction address are moved to the memory location whose address is one less than the content of register SP. The low-order eight bits of the next instruction address are moved to the memory location whose address is two less than the content of register SP. The content of register SP is decremented by 2. Control is transferred to the instruction whose address is specified in byte 3 and byte 2 of the current instruction.



Cycles: 5
States: 17
Addressing: immediate/reg. indirect
Flags: none

Ccondition addr (Condition call)

If (CCC),

((SP) – 1) ← (PCH)

((SP – 2) ← (PCL)

(SP) ← (SP) – 2

(PC) ← (byte 3) (byte 2)

If the specified condition is true, the actions specified in the CALL instruction (see above) are performed; otherwise, control continues sequentially.

1	1	C	C	C	1	0	0
low-order addr							
high-order addr							

Cycles:	3/5
States:	11/17
Addressing:	immediate/reg. indirect
Flags:	none

RET (Return)

(PCL) ← ((SP));

(PCH) ← ((SP) + 1);

(SP) ← (SP) + 2);

The content of the memory location whose address is specified in register SP is moved to the low-order eight bits of register PC. The content of the memory location whose address is one more than the content of the register SP is moved to the high-order eight bits of register PC. The content of register SP is incremented by 2.

1	1	0	0	1	0	0	1

Cycles:	3
States:	10
Addressing:	reg. indirect
Flags:	none

Rcondition (Conditional return)

If (CCC),
(PCL) ← ((SP))
(PCH) ← ((SP) + 1)
(SP) ← (SP) + 2

If the specified condition is true, the actions specified in the RET instruction (see above) are performed; otherwise, control continues sequentially.

1	1	C	C	C	0	0	0

Cycles:	1/3
States:	5/11
Addressing:	reg. indirect
Flags:	none

RST n (Restart)

((SP) - 1) ← (PCH)
((SP) - 2) ← (PCL)
(SP) ← (SP) - 2
(PC) ← 8 * (NNN)

The high-order eight bits of the next instruction address are moved to the memory location whose address is one less than the content of register SP. The low-order eight bits of the next instruction address are moved to the memory location whose address is two less than the content of register SP. The content of register SP is decremented by two. Control is transferred to the instruction whose address is eight times the content of NNN.

1	1	N	N	N	1	1	1

Cycles:	3
States:	11
Addressing:	reg. indirect
Flags:	none

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	N	N	N	0	0	0

Program Counter After Restart

PCHL (Jump H and L indirect—move H and L to PC)

$(PCH) \leftarrow (H)$
$(PCL) \leftarrow (L)$

The content of register H is moved to the high-order eight bits of register PC. The content of register L is moved to the low-order eight bits of register PC.

1	1	1	0	1	0	0	1

Cycles:	1
States:	5
Addressing:	register
Flags:	none

Stack, I/O, and Machine Control Group

This group of instructions performs I/O, manipulates the Stack, and alters internal control flags.

Unless otherwise specified, **condition flags are not affected by any instructions in this group.**

PUSH rp (Push)

$((SP) - 1) \leftarrow (rh)$
$((SP) - 2) \leftarrow (rl)$
$(SP) \leftarrow (SP) - 2$

The content of the high-order register of register pair rp is moved to the memory location whose address is one less than the content of register SP. The content of the low-order registerof register pair rp is moved to the memory location whose address is two less than the content of register SP. The content of register SP is decremented by 2. **Note: Register pair rp = SP may not be specified.**

1	1	R	P	0	1	0	1

Cycles:	3
States:	11
Addressing:	reg. indirect
Flags:	none

PUSH PSW (Push processor status word)

$((SP) - 1) \leftarrow (A)$
$((SP) - 2)_0 \leftarrow (CY), ((SP) - 2)_1 \leftarrow 1$
$((SP) - 2)_2 \leftarrow (P), ((SP) - 2)_3 \leftarrow 0$
$((SP) - 2)_4 \leftarrow (AC), ((SP) - 2)_5 \leftarrow 0$
$((SP) - 2)_6 \leftarrow (Z), ((SP) - 2)_7 \leftarrow (S)$
$(SP) \leftarrow (SP) - 2$

The content of register A is moved to the memory location whose address is one less than register SP. The contents of the condition flags are assembled into a processor status word and the word is moved to the memory location whose address is two less than the content of register SP. The content of register SP is decremented by two.

1	1	1	1	0	1	0	1

Cycles:	3
States:	11
Addressing:	reg. indirect
Flags:	none

FLAG WORD

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z	0	AC	0	P	1	CY

POP rp (Pop)

$(rl) \leftarrow ((SP))$
$(rh) \leftarrow ((SP)) + 1)$
$(SP) \leftarrow (SP) + 2$

The content of the memory location, whose address is specified by the content of register SP, is moved to the low-order register of register pair rp. The content of the memory location, whose address is one more than the content of register SP, is moved to the high-order register of register pair rp. The content of register SP is incremented by 2. **Note: Register pair rp = SP may not be specified.**

1	1	R	P	0	0	0	1

Cycles:	3
States:	10
Addressing:	reg. indirect
Flags:	none

POP PSW (Pop processor status word)

$(CY) \leftarrow ((SP))_0$
$(P) \leftarrow ((SP))_2$
$(AC) \leftarrow ((SP))_4$
$(Z) \leftarrow ((SP))_6$
$(S) \leftarrow ((SP))_7$
$(A) \leftarrow ((SP) + 1)$
$(SP) \leftarrow (SP) + 2$

The content of the memory location whose address is specified by the content of register SP is used to restore the condition flags. The content of the memory location whose address is one more than the content of register SP is moved to register A. The content of register SP is incremented by 2.

1	1	1	1	0	0	0	1

Cycles:	3
States:	10
Addressing:	reg. indirect
Flags:	Z,S,P,CY,AC

XTHL (Exchange stack top with H and L)

$(L) \longleftrightarrow ((SP))$
$(H) \longleftrightarrow ((SP) + 1)$

The content of the L register is exchanged with the content of the memory location whose address is specified by the content of register SP. The content of the H register is exchanged with the content of the memory location whose address is one more than the content of register SP.

1	1	1	0	0	0	1	1

Cycles:	5
States:	18
Addressing:	reg. indirect
Flags:	none

SPHL (Move HL to SP)

$(SP) \leftarrow (H) (L)$

The contents of registers H and L (16 bits) are moved to register SP.

1	1	1	1	1	0	0	1

Cycles:	1
States:	5
Addressing:	register
Flags:	none

IN port (Input)

(A) ← (data)

The data placed on the eight bit bi-directional data bus by the specified port is moved to register A.

1	1	0	1	1	0	1	1
port							

Cycles: 3
States: 10
Addressing: direct
Flags: none

OUT port (Output)

(data) ← (A)

The content of register A is placed on the eight bit bi-directional data bus for transmission to the specified port.

1	1	0	1	0	0	1	1
port							

Cycles: 3
States: 10
Addressing: direct
Flags: none

EI (Enable interrupts)

The interrupt system is enabled **following the execution of the next instruction.**

1	1	1	1	1	0	1	1

Cycles: 1
States: 4
Flags: none

DI (Disable interrupts)

The interrupt system is disabled **immediately following the execution of the DI instruction.**

1	1	1	1	0	0	1	1

Cycles: 1
States: 4
Flags: none

HLT (Halt)

The processor is stopped. The registers and flags are unaffected.

0	1	1	1	0	1	1	0

Cycles: 1
States: 7
Flags: none

NOP (No op)

No operation is performed. The registers and flags are unaffected.

0	0	0	0	0	0	0	0

Cycles: 1
States: 4
Flags: none

8080 Instruction Set

Summary of Processor Instructions

		Instruction Code[1]								Clock[2]
Mnemonic	Description	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Cycles
MOVE, LOAD, AND STORE										
MOVr1,r2	Move register to register	0	1	D	D	D	S	S	S	5
MOV M,r	Move register to memory	0	1	1	1	0	S	S	S	7
MOV r,M	Move memory to register	0	1	D	D	D	1	1	0	7
MVI r	Move immediate register	0	0	D	D	D	1	1	0	7
MVI M	Move immediate memory	0	0	1	1	0	1	1	0	10
LXI B	Load immediate register Pair B & C	0	0	0	0	0	0	0	1	10
LXI D	Load immediate register Pair D & E	0	0	0	1	0	0	0	1	10
LXI H	Load immediate register Pair H & L	0	0	1	0	0	0	0	1	10
STAX B	Store A indirect	0	0	0	0	0	0	1	0	7
STAX D	Store A indirect	0	0	0	1	0	0	1	0	7
LDAX B	Load A indirect	0	0	0	0	1	0	1	0	7
LDAX D	Load A indirect	0	0	0	1	1	0	1	0	7
STA	Store A direct	0	0	1	1	0	0	1	0	13
LDA	Load A direct	0	0	1	1	1	0	1	0	13
SHLD	Store H & L direct	0	0	1	0	0	0	1	0	16
LHLD	Load H & L direct	0	0	1	0	1	0	1	0	16
XCHG	Exchange D & E, H & L Registers	1	1	1	0	1	0	1	1	4
STACK OPS										
PUSH B	Push register Pair B & C on stack	1	1	0	0	0	1	0	1	11
PUSH D	Push register Pair D & E on stack	1	1	0	1	0	1	0	1	11
PUSH H	Push register Pair H & L on stack	1	1	1	0	0	1	0	1	11
PUSH PSW	Push A and Flags on stack	1	1	1	1	0	1	0	1	11
POP B	Pop register Pair B & C off stack	1	1	0	0	0	0	0	1	10
POP D	Pop register Pair D & E off stack	1	1	0	1	0	0	0	1	10
POP H	Pop register Pair H & L off stack	1	1	1	0	0	0	0	1	10
POP PSW	Pop A and Flags off stack	1	1	1	1	0	0	0	1	10
XTHL	Exchange top of stack, H & L	1	1	1	0	0	0	1	1	18
SPHL	H & L to stack pointer	1	1	1	1	1	0	0	1	5
LXI SP	Load immediate stack pointer	0	0	1	1	0	0	0	1	10
INX SP	Increment stack pointer	0	0	1	1	0	0	1	1	5
DCX SP	Decrement stack pointer	0	0	1	1	1	0	1	1	5

Notes:
1. DDD or SSS: B 000, C 001, D 010, E 011, H 100, L 101, Memory 110, A 111.
2. Two possible cycle times. (6/12) indicate instruction cycle dependent on condition flags.

Mnemonic	Description	Instruction Code[1] D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Clock[2] Cycles
JUMP										
JMP	Jump unconditional	1	1	0	0	0	0	1	1	10
JC	Jump on carry	1	1	0	1	1	0	1	0	10
JNC	Jump on no carry	1	1	0	1	0	0	1	0	10
JZ	Jump on zero	1	1	0	0	1	0	1	0	10
JNZ	Jump on no zero	1	1	0	0	0	0	1	0	10
JP	Jump on positive	1	1	1	1	0	0	1	0	10
JM	Jump on minus	1	1	1	1	1	0	1	0	10
JPE	Jump on parity even	1	1	1	0	1	0	1	0	10
JPO	Jump on parity odd	1	1	1	0	0	0	1	0	10
PCHL	H & L to program counter	1	1	1	0	1	0	0	1	5
CALL										
CALL	Call unconditional	1	1	0	0	1	1	0	1	17
CC	Call on carry	1	1	0	1	1	1	0	0	11/17
CNC	Call on no carry	1	1	0	1	0	1	0	0	11/17
CZ	Call on zero	1	1	0	0	1	1	0	0	11/17
CNZ	Call on no zero	1	1	0	0	0	1	0	0	11/17
CP	Call on positive	1	1	1	1	0	1	0	0	11/17
CM	Call on minus	1	1	1	1	1	1	0	0	11/17
CPE	Call on parity even	1	1	1	0	1	1	0	0	11/17
CPO	Call on parity odd	1	1	1	0	0	1	0	0	11/17
RETURN										
RET	Return	1	1	0	0	1	0	0	1	10
RC	Return on carry	1	1	0	1	1	0	0	0	5/11
RNC	Return on no carry	1	1	0	1	0	0	0	0	5/11
RZ	Return on zero	1	1	0	0	1	0	0	0	5/11
RNZ	Return on no zero	1	1	0	0	0	0	0	0	5/11
RP	Return on positive	1	1	1	1	0	0	0	0	5/11
RM	Return on minus	1	1	1	1	1	0	0	0	5/11
RPE	Return on parity even	1	1	1	0	1	0	0	0	5/11
RPO	Return on parity odd	1	1	1	0	0	0	0	0	5/11
RESTART										
RST	Restart	1	1	A	A	A	1	1	1	11
INCREMENT AND DECREMENT										
INR r	Increment register	0	0	D	D	D	1	0	0	5
DCR r	Decrement register	0	0	D	D	D	1	0	1	5
INR M	Increment memory	0	0	1	1	0	1	0	0	10
DCR M	Decrement memory	0	0	1	1	0	1	0	1	10

Notes:
1. DDD or SSS: B 000, C 001, D 010, E 011, H 100, L 101, Memory 110, A 111.
2. Two possible cycle times. (6/12) indicate instruction cycles dependent on condition flags.

Mnemonic	Description	Instruction Code[1] D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Clock[2] Cycles
INX B	Increment B & C registers	0	0	0	0	0	0	1	1	5
INX D	Increment D & E registers	0	0	0	1	0	0	1	1	5
INX H	Increment H & L registers	0	0	1	0	0	0	1	1	5
DCX B	Decrement B & C	0	0	0	0	1	0	1	1	5
DCX D	Decrement D & E	0	0	0	1	1	0	1	1	5
DCX H	Decrement H & L	0	0	1	0	1	0	1	1	5
ADD										
ADD r	Add register to A	1	0	0	0	0	S	S	S	4
ADC r	Add register to A with carry	1	0	0	0	1	S	S	S	4
ADD M	Add memory to A	1	0	0	0	0	1	1	0	7
ADC M	Add memory to A with carry	1	0	0	0	1	1	1	0	7
ADI	Add immediate to A	1	1	0	0	0	1	1	0	7
ACI	Add immediate to A with carry	1	1	0	0	1	1	1	0	7
DAD B	Add B & C to H & L	0	0	0	0	1	0	0	1	10
DAD D	Add D & E to H & L	0	0	0	1	1	0	0	1	10
DAD H	Add H & L to H & L	0	0	1	0	1	0	0	1	10
DAD SP	Add stack pointer to H & L	0	0	1	1	1	0	0	1	10
SUBTRACT										
SUB r	Subtract register from A	1	0	0	1	0	S	S	S	4
SBB r	Subtract register from A with borrow	1	0	0	1	1	S	S	S	4
SUB M	Subtract memory from A	1	0	0	1	0	1	1	0	7
SBB M	Subtract memory from A with borrow	1	0	0	1	1	1	1	0	7
SUI	Subtract immediate from A	1	1	0	1	0	1	1	0	7
SBI	Subtract immediate from A with borrow	1	1	0	1	1	1	1	0	7
LOGICAL										
ANA r	And register with A	1	0	1	0	0	S	S	S	4
XRA r	Exclusive Or register with A	1	0	1	0	1	S	S	S	4
ORA r	Or register with A	1	0	1	1	0	S	S	S	4
CMP r	Compare register with A	1	0	1	1	1	S	S	S	4
ANA M	And memory with A	1	0	1	0	0	1	1	0	7
XRA M	Exclusive Or memory with A	1	0	1	0	1	1	1	0	7
ORA M	Or memory with A	1	0	1	1	0	1	1	0	7
CMP M	Compare memory with A	1	0	1	1	1	1	1	0	7
ANI	And immediate with A	1	1	1	0	0	1	1	0	7
XRI	Exclusive Or immediate with A	1	1	1	0	1	1	1	0	7

Notes:
1. DDD or SSS: B 000, C 001, D 010, E 011, H 100, L 101, Memory 110, A 111.
2. Two possible cycle times. (6/12) indicate instruction cycles dependent on condition flags.

Mnemonic	Description	Instruction Code[1] D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Clock[2] Cycles
ORI	Or immediate with A	1	1	1	1	0	1	1	0	7
CPI	Compare immediate with A	1	1	1	1	1	1	1	0	7
ROTATE										
RLC	Rotate A left	0	0	0	0	0	1	1	1	4
RRC	Rotate A right	0	0	0	0	1	1	1	1	4
RAL	Rotate A left through carry	0	0	0	1	0	1	1	1	4
RAR	Rotate A right through carry	0	0	0	1	1	1	1	1	4
SPECIALS										
CMA	Complement A	0	0	1	0	1	1	1	1	4
STC	Set carry	0	0	1	1	0	1	1	1	4
CMC	Complement carry	0	0	1	1	1	1	1	1	4
DAA	Decimal adjust A	0	0	1	0	0	1	1	1	4
INPUT/OUTPUT										
IN	Input	1	1	0	1	1	0	1	1	10
OUT	Output	1	1	0	1	0	0	1	1	10
CONTROL										
EI	Enable interrupts	1	1	1	1	1	0	1	1	4
DI	Disable interrupt	1	1	1	1	0	0	1	1	4
NOP	No-operation	0	0	0	0	0	0	0	0	4
HLT	Halt	0	1	1	1	0	1	1	0	7

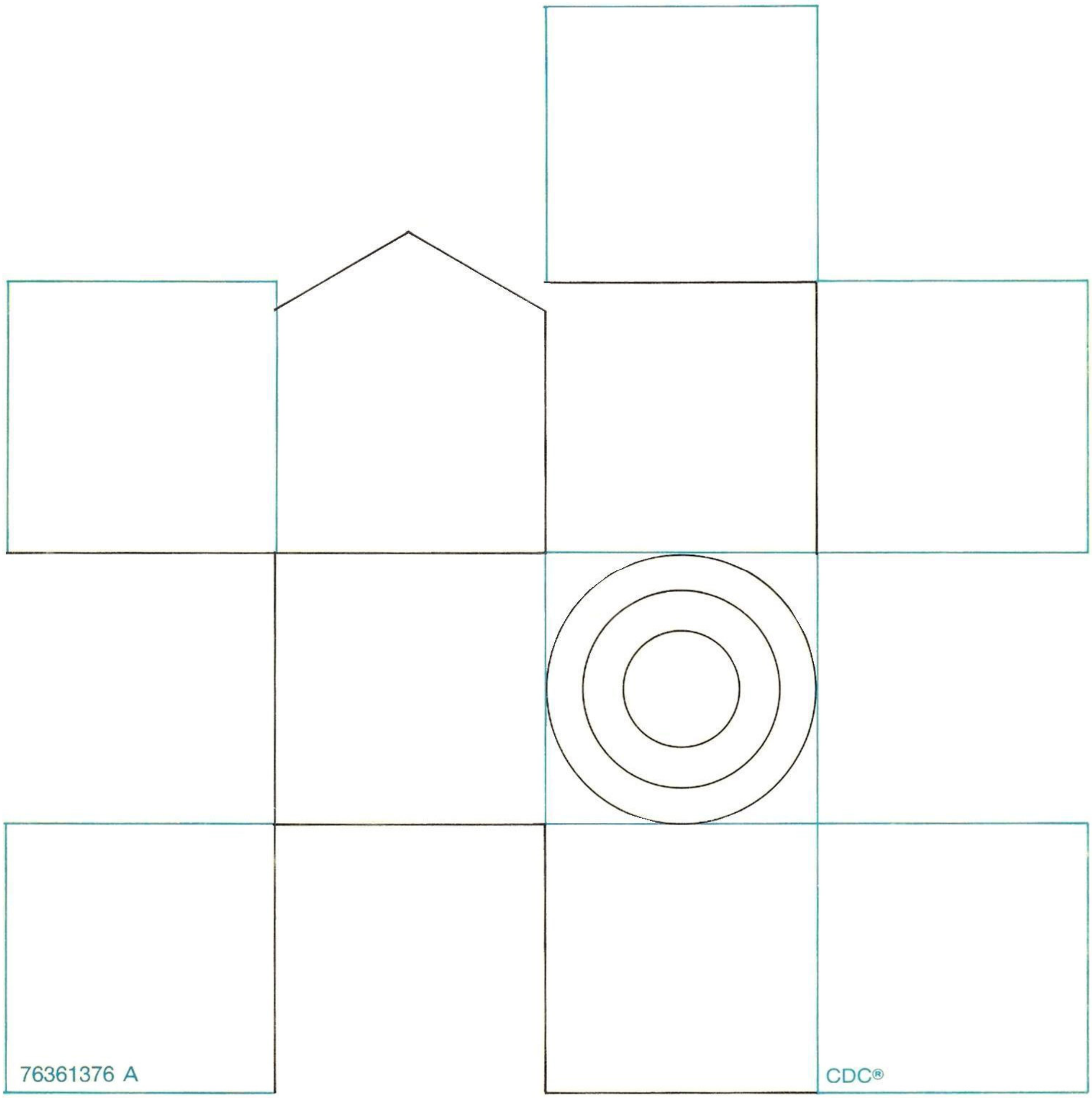
Notes:
1. DDD or SSS: B 000, C 001, D 010, E 011, H 100, L 101, Memory 110, A 111.
2. Two possible cycle times. (6/12) indicate instruction cycles dependent on condition flags.

(Blank Page)

(Blank Page)

(Blank Page)

(Blank Page)



76361376 A

CDC®