UNIVERSAL TRAINER

LAB MANUAL

for
Board Revisions 1 and 2

MANUAL REVISION 1.1
February 1993

EMAC, inc.

EQUIPMENT MONITOR AND

CONTROL

CARBONDALE, IL 62901
618-529-4525

LAB #1
INTRODUCTION TO COMPUTERS

INTRODUCTION

A computer is an electronic device that follows instructions in order to do useful work. There are many different kinds of computers. The computers used by the IRS for instance, require a great deal of computing power in order to process tax returns for the millions of people in the United States. These computers are quite large and cost millions of dollars. The desktop computer on the other hand is not as powerful as the IRS computer, but then again it is quit smaller and doesn't cost near as much. The desktop computers size is made possible by the use of microprocessors. Microprocessors are small electronic chips (integrated circuits) that contain at least 75% of the computing power of the digital computer. Besides reducing the size of a computer, microprocessors are also responsible for reducing their cost. The small size and low cost of microprocessors has facilitated their use in almost every electronic device from hospital equipment to military weapons, from toys to home appliances.

All computers regardless of their size and power require some way to input and output data in order to be useful. A desktop computer for example receives input commands from the keyboard and displays output on the computer monitor. A modern microwave oven which also contains a microprocessor receives input commands from the keypad and the output is seen on the calculator type display. Their are however, other inputs and outputs not so easily identified on the microwave oven. There is an input switch on the door that turns the light on and the oven off. There is an output control line that turns the oven on and off in response to the various inputs of the system.

Memory is another required component of a computer. Memory is used to store computer instructions (programs) and data. Again in the example of the microwave oven, the program is what controls the oven and allows you to enter the amount of time (the data) to cook your food.

Computers basically perform three functions: inputing data, processing the data and outputing data. Some devices that are commonly used to input information to a computer are keyboards, dip switches and joysticks. Common computer output devices are light emitting diodes (LEDs), liquid crystal displays (LCDs), video monitors, and printers. Disk drives and modems are common devices that have both input and output characteristics.

A microprocessor, which is also referred to as the central processing unit (CPU), processes the information that is input to the computer and determines what data will be sent to the output devices. The microprocessor has within it an arithmetic logic unit (ALU) which performs addition, subtraction, comparisons and logical functions, which will be discussed later. One thing you should know about computers is that they don't really think like people do. They can only do what the computer manufacturer or the computer user tells them to do. The microprocessor can do many different things, but in order for something useful to be done it must follow a group of instructions called a program. Below is a "program" or a group of instructions that you could write which a person may follow in order to quench their thirst.

1) Get a glass.
2) Get some milk from the refrigerator.
3) Pour the milk in the glass.
4) Drink the milk.
5) If you are still thirsty continue from step three.
6) Put glass in sink
7) put milk in refrigerator.

In the same way, by knowing the microprocessor's language, you can give it a group of instructions that it can perform. The microprocessor would perform these instructions exactly as you commanded it. The microprocessor can only obey one instruction of a program at a time and these instructions tell the microprocessor whether to input data, output data or perform one of the ALU functions.

REFERENCE

The Intelligent Microcomputer by Goody, Chapters 1, 2, and 3.

OBJECTIVES

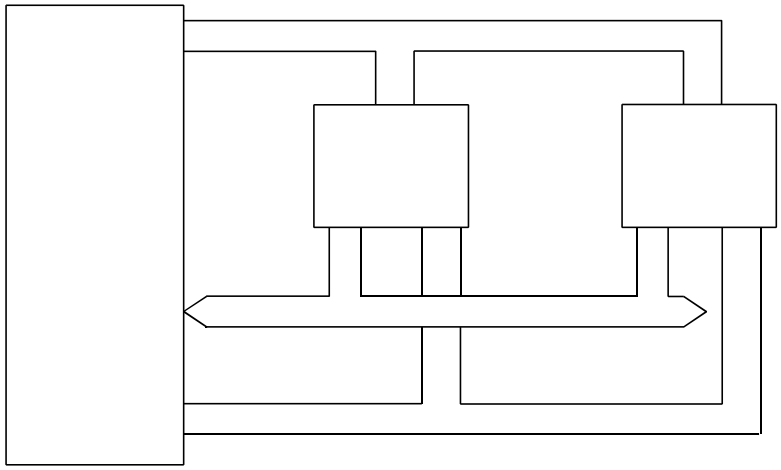
* To understand the building blocks of the computer.
* To define the various types of memory.
* To define Input/Output (I/O) as it is used on the 8085 microprocessor.
* To define the internal structure of the 8085 microprocessor.

PROCEDURE

The 8085 microprocessor with the addition of a few components is a fully functioning computer system. The microprocessor is the brains of the computer, and it executes the instructions that make up a computer program. These instructions are held in memory where the microprocessor accesses them as required. Each instruction is placed in a different location of memory with its own unique address, much the same as each house on a street has its own address. The address is placed on the address bus where it is sent to the memory. The memory then places the instruction at that address on the data bus where it is sent to the microprocessor. This is basically how a computer executes instructions. The address bus always carries the address from the microprocessor to the memory or I/O device. The data bus on the other hand is used to send information both from and to the microprocessor and therefore is called a bidirectional bus. Another bus present in the microprocessor system is the control bus. This bus basically consists of the control lines which keep the microprocessor operational. For example whether to read or write from memory, do an I/O operation, wait till the memory is ready, etc.

MEMORY

The 8085 microprocessor can access 65536 individual memory locations in the range 0 to 65535 in decimal (0 to FFFF in hex) but only one at a time. There are two types of memory in most microprocessor based systems, memory that can be read but not written to, which is called Read Only Memory (ROM), and memory that can be read from and written to, which is called Read/Alter Memory or Random Access Memory (RAM).
ROM memory retains its contents when power is removed. Memory that retains its contents when power is removed is called nonvolatile memory. RAM memory on the other hand loses all of its contents when power is removed and is referred to as volatile memory. Adding a battery to a RAM chip can change it from a volatile memory to a nonvolatile memory.



MICROCOMPUTER BLOCK DIAGRAM
FIG. 1.0

Both RAM and ROM chips have address pins which are connected to the microprocessor's address pins. These pins are connected through what is called an address bus and through this bus the microprocessor can select a memory location for writing or reading of data. Writing to ROM chips has no effect since it is read only memory. The RAM and ROM chips in the Universal Trainer have eight data pins which send data to, or receive data from the microprocessor through a group of eight connections called the data bus. Since there are 8 data pins on the RAM and ROM chips, this allows numbers from 0 to 255 (0 to FF in hex) to be read from or written to each memory location.

INPUT/OUTPUT

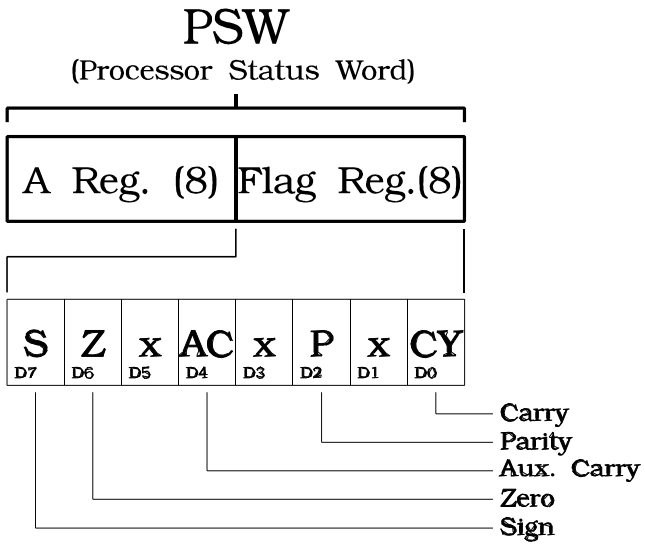
The 8085 microprocessor can also send data to and receive data from chips other than the RAM or ROM. When the microprocessor wants to perform input or output it disables the RAM and ROM chips and sends an input/output (I/O) address to the address bus. The I/O address is only 8 bits but it appears on the lower 8 bits (A0-A7) and the higher 8 bits (A8-A15) of the address bus simultaneously. Since the address generated is only 8 bits long, only I/O addresses from 0-255 (0-FF hex) can be selected. Most microprocessor-based systems have circuitry which decode the address from the address bus and select the appropriate I/O device. Usually these devices are dedicated to either input only or output only. If an input device has been selected, 8 bits of data is transmitted from the input device to the data bus and into the microprocessor. If an output device has been selected, the 8 bits of data is sent from the microprocessor to the data bus and to the output device.

The control bus is a group of connections which provide control over reading or writing of memory or I/O devices. Figure 1.0 is a block diagram showing the way the CPU (microprocessor) connects to the memory and I/O devices through the address bus, data bus and control bus.

REGISTERS

The 8085 microprocessor has within it temporary storage devices called registers. Registers work similar to RAM in that they store binary values. The 8 bit general purpose registers provided by the 8085 are named A, B, C, D, E, H and L. The A register is often referred to as "the accumulator". The A register in addition to being used as a general purpose register has some unique abilities. The A register is always used for all I/O instructions, logic instructions, and most arithmetic instructions. For example if you subtract two numbers the difference would be located in the A register.

The 8085 has many instructions which use these individual general purpose registers. There are also instructions which view a pair of the general purpose registers as a single 16 bit register. The register pairs that are used in these instructions are BC, DE, and HL. When they are paired with other registers C,E and L represent the least significant 8 bits of the register pairs (bits 0-7) and B,D and H represent the most significant 8 bits (bits 8-15). Some instructions view the A register and flag register (described below) as a 16 bit register called the processor status word (PSW). The A register is the most significant 8 bits and the flag register is the least significant 8 bits of register. Figure 1.1 shows the PSW with a diagram of the individual bits of the flag register.



x : Undefined

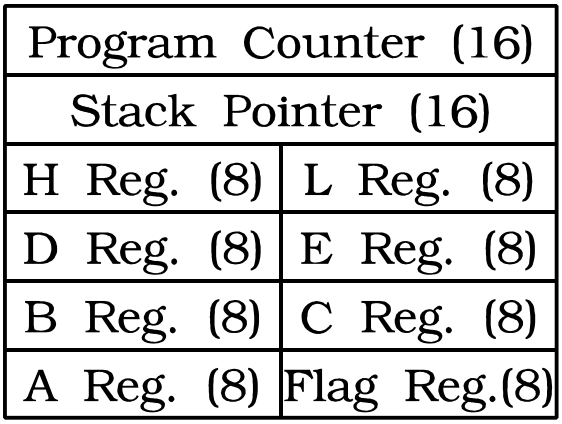
FIG. 1.1

There are also registers that are dedicated to special purposes. These registers and their descriptions are as follows:

The stack pointer (SP) is a 16 bit register which points to a memory location in RAM which will hold temporary values in an area of RAM called the stack. The stack is explained in chapter 15 of The Intelligent Microcomputer by Goody.

The program counter (PC) is a 16 bit register which points to the memory location of the next machine language instruction to be executed.

The flag register is an 8 bit register which has individual bits, called flags, that indicate the result of arithmetic or logical operations. The flags are covered in later labs.



8085 REGISTERS
FIG. 1.2

QUESTIONS 1.0

1. Define microprocessor.

2. Why does a computer need I/O?

3. What type of memory can only be read from?

4. What type of memory loses the information contained in it when power is removed?

5. Is the address bus unidirectional or bidirectional?

6. Is the data bus unidirectional or bidirectional?

7. How many memory locations can the 8085 access?

8. How many I/O devices can the 8085 access?

9. What is another name for the A register?

10. What makes the A register different from the other general purpose registers?

11. How many bits are contained in the A register?

12. How many bits are contained in a register pair?

13. How many bits are contained in the stack pointer?

14. How many different flags are available in the 8085?

15. What does PSW stand for?

16. When referring to the B/C register pair which register holds the 8 bits of low order data?

LAB #2
COMPUTER NUMBER SYSTEMS

INTRODUCTION

In order to be able to program a computer, its necessary to be able to speak in a form that the computer can understand. Computers being digital devices (using two states) use the binary number system as their native tongue. The hexadecimal number system, like binary is a power of two number system also. Each hexadecimal digit is represented by 16 different numeric symbols verses 2 for a binary digit. Hexadecimal therefore is more efficient then using binary when listing computer codes and since hexadecimal is a power of 2 number system conversion to the binary number system is a simple procedure.
Since we humans use the decimal number system, converting from decimal to binary (the number system computers understand) is necessary in order to program computers at a low level.

OBJECTIVES

* Define a bit.
* Define a byte.
* Define a word.
* Define binary code.
* Define hexadecimal code.
* Convert a binary number to decimal.
* Convert a binary number to hexadecimal.
* Convert a hexadecimal number to binary.
* Define least significant and most significant.

PROCEDURE

A microprocessor performs all arithmetic in binary, although it may be translated to different forms (i.e. decimal, hexadecimal..). The binary number system consists of the numbers 0 and 1 which are called binary digits or bits for short. There are several words used to represent the binary numbers 0 and 1 and they are often used interchangeably, depending on the context in which they are used. They are as follows:

binary 1 = true on high set +5 volts set

binary 0 = false off low reset 0 volts clear

In the 8085 microprocessor, binary numbers are organized in groups of 8 bits which are called bytes and groups of 16 bits which are called words. When referring to a byte, it is often necessary to describe particular bits, so the numbering of each of the 8 bits is as follows:

7 6 5 4 3 2 1 0

So in the binary number, 10001000, bit 7 and bit 3 are 1 and the rest are 0. In binary number 00010001, bit 4 and bit 0 are have a value of 1 and the rest are 0. When referring to a word, the bits are numbered:

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

As you know, in decimal numbers, each position of a digit has a different weight. For example in the decimal number 1732:
(The digit numbers are read from right to left, 0 to 3)

digit #	weight		value	= decimal result
3	10^3	*	1	= 1000
2	10^2	*	7	= 700
1	10^1	*	3	= 30
0	10^0	*	2	= 2
sum of numbers * weights				**= 1732**

In the same way the binary system has weights for each bit position. The weight for a bit is 2 to the power of the bit number ($2^{(\text{bit number})}$). The binary number 11111111 can be converted to decimal by knowing the weights of each bit, as in the example below:

bit #	weight		value	= decimal result
7	2^7	*	1	= 128
6	2^6	*	1	= 64
5	2^5	*	1	= 32
4	2^4	*	1	= 16
3	2^3	*	1	= 8
2	2^2	*	1	= 4
1	2^1	*	1	= 2
0	2^0	*	1	= 1
sum of bits * weights				**= 255**

If one of the bit positions had been a zero, that bit position's weight would not have been added to the sum.

To convert a decimal number to another base repeatedly divide the number by the base until the quotient equals zero. For example to find the binary (base 2) equivalent of the decimal number 58 we proceed as follows:

```
58 / 2 = 29 with a remainder of     0 (LSB)
29 / 2 = 14 with a remainder of     1
14 / 2 = 7  with a remainder of     0
07 / 2 = 3  with a remainder of     1
03 / 2 = 1  with a remainder of     1
01 / 2 = 0  with a remainder of     1 (MSB)
```

The binary equivalent of the decimal number 58 is therefore: 111010. The binary number 111010 can be varified by converting it back to decimal. Conversion from decimal to hexidecimal can be accomplished by dividing the decimal number by 16 instead instead of 2. The remainder when dividing by 16 will range from 0 to 15, or 0 to F in hex (see below).

The three forms of numbers we will use in this manual are: binary, hexadecimal (hex, for short) and decimal. Below is a table of the binary, and hex equivalents of the decimal numbers 0 through 20.

DECIMAL	HEX	BINARY
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110

15	F	1111
16	10	10000
17	11	10001
18	12	10010
19	13	10011
20	14	10100

Hexadecimal is used to represent binary values because it is more efficient then binary and it is very easy to convert numbers from binary to hexadecimal. For example, to convert the following binary number to hexadecimal:

```
101101101101011
```

Start from the right digit and put the number into groups of four binary digits (bits). If there are not enough bits in the number to make a full four bits in the group on the left side, add zeros to the left of the number.

```
0101 1011 0110 1011
```

Now replace the binary groups with their hexadecimal equivalents using the table above and you will get the following result:

```
5   B   6   B
```

It is just as easy to convert hex to binary. Merely replace each hex digit with the corresponding 4 binary digits from the table above and you have your binary number, for example:

```
HEX
F   C   1   8

BINARY
1111 1100 0001 1000
```

In this manual you will see the words "least significant" or "low order" and "most significant" or "high order". These refer to the mathematical weight of the part of a number that is being described. In all number systems the digit on the left end is the most significant or high order digit and the digit on the right end is the least significant or low order digit. For example in the binary number 00010010, the bit on the left end is the most significant bit (MSB) and the bit on the right end is the least significant bit (LSB). In the hex word 01FF the left two digits are the most significant byte and the two right digits are the least significant byte.

QUESTIONS 2.0

1. What two numbers represents a binary digit?

2. Give an example of non-numeric names for the two numbers of question 1.

3. How many bits are in a byte?

4. How many bits in a word?

5. Give the decimal equivalent of the binary number 10010110.

6. Give the hex equivalent of the binary number 10010110.

7. Give the binary equivalent of the decimal number 115.

8. Give the hex equivalent of the decimal number 307.

9. Give the binary equivalent of the hex number D6.

10. Explain what is meant when hex is said to be more efficient for the user than binary.

LAB #3
COMPUTER LOGIC OPERATIONS

INTRODUCTION

Computers being digital systems, are programmed using digital logic instructions. The hardware of a computer consists of logic gates which in conjunction with the software determines what the computer does. A special group of instructions called logic instructions, lets computer programs make decisions much the same as the logic gates of a computer make decisions needed for the computer to operate. As a matter of fact a computer can simulate in software, the operation of hardware logic gates.

If a computer program allows a light to come on if one of two switches is in the ON position, it is using digital logic to accomplish the operation. Other possibilities include the light only being turned on if both the switches are in the ON position. Logic instructions also allow us to isolate a particular bit position or positions. Using logic instructions a bit can be turned on, turned off, or turned to the opposite state, while not disturbing other bit positions. If each bit position is visualized as being connected to a light, a alarm, a motor, etc. with a 1 turning the particular device on and a 0 turning it off, it becomes apparent that being able to control each bit position individually is an important feature.

REFERENCE

The Intelligent Microcomputer by Goody, Chapter 12.

OBJECTIVES

* Define the AND operation.
* Define the OR operation.
* Define the EXCLUSIVE OR (XOR) operation.
* Define the NOT operation.
* Define mask.

PROCEDURE

The 8085 supports 4 logical operations:

1) The AND operation takes two input bits and returns a 1 bit if both input bits are 1 and a 0 bit if either bit is 0.

2) The OR operation takes two input bits and returns a 1 bit if either input bit is 1 and a 0 bit if both input bits are 0.

3) The XOR operation takes two input bits and returns a 0 bit if the input bits are the same and a 1 bit if they are different.

4) The NOT operation takes one input bit and returns a 1 if the input bit is 0 and returns a 0 if the input bit is 1. This is called complementing, inverting, or toggling the bit.

The 8085 performs these operations 8 bits at a time, by performing the logic operation on each bit position, For example:

```
        01110010 <-X
AND     10010011 <-Y
        00010010 <-Z
```

Bit 0 of X is ANDed with bit 0 of Y and gives the result in bit 0 of Z. Bit 1 of X is ANDed with bit 1 of Y and gives the result in bit 1 of Z and the pattern continues on up to bit 7.

The following shows the way logical operations work with bytes:

```
  00010010          01100111          11101101
AND 01000010    OR  10101110    XOR   01111001      NOT 11001010
                    11101111          10010100          00110101
00000010
```

If we envision each bit position as being connected to a different light in a house, using logic operations we can control each light individually or in groups. The trick is to control a particular light without knowing the current state of each light and not upsetting the other lights in the house. For example to turn on light #2, a one must be in bit position two. To turn off light #2, a zero must be in bit position two. We would like to turn light #2 on and off without disturbing any of the other seven lights in the house. For the following example the arbitrarily chosen byte below represents the current state of eight different lights in a house.

```
01001010
```

This byte tells us that light #6 is on, light #3 is on, light #1 is on, and all the other lights are off. To turn light #2 on we use a mask value. A mask is a value used to manipulate different bit positions. This value when used with the appropriate logic operation performs the function we desire. The mask value chosen for the example is:

```
00000100
```

This mask value when ORed with the current state of the lights will force light #2 on.

```
  01001010
OR 00000100
   01001110
```

Notice that all the lights that were on are still on, and all the lights that were off are still off, with the exception of light #2. If later we decide to turn light #2 off, we use a different mask value and a different logic operation. Finding the correct mask value for turning the light off is not as simple as it was for turning the light on. First we must take the complement using the NOT operation, of the mask we used to turn the light on, then we AND this value with current state of the lights in order to turn the light off. This translates into simply placing a zero in any bit positions to be turned off and a one in all other bit positions.

```
NOT 00000100
    11111011
```

The complementing of the mask used to turn the light on, gives us the new mask value needed to turn the light off. This new mask value has a zero in each bit position to be turned off, and a one in all other bit positions.

```
  01001110
AND 11111011
    01001010
```

ANDing the new mask value with current state of the lights turns light #2 off, while not affecting any of the other lights. The current state of the lights is now the same as it was before we turned light #2 on. By adding more ones to the mask value, more then one light can be turned on, and by adding more zeros, more then one light can be turned off. Remember to use the OR operator to turn things on and the AND operator to turn things off.

To toggle or compliment a bit position the exclusive OR (XOR) logic operator is used. This allows a bit to be switched from a high to a low or from a low to a high without knowing the current state of the bit position. The mask value should contain a 1 in each bit poition that requires toggling and 0's in all other bit positions. Like the other logic operations this only affects the selected bit positions. All positions that contain a 0 will retain their original values. For example if we wish to toggle bits 0 and 6 of the result of

the above AND operation, the mask value would be 01000001 and yeild a result of:

```
        01001010
XOR     01000001
        --------
        00001011
```

Notice that only bit positions 1 and 6 have changed, all other bit positions have retained the value they had before the XOR operation took place.

QUESTIONS 3.0

1. Complete the following logic operations:

```
        11100110        10111011        00110110
AND     00110101    OR  01101001    XOR 10011010    NOT 11101011
```

2. Define the term mask.

3. Give the 8 bit mask value and the logic operation to turn on bit #3 and bit #5 without disturbing any of the other bits.

4. Give the 8 bit mask value and the logic operation to turn off bit #1 and bit #4 without disturbing any of the other bits.

5. Give the 8 bit mask value using the XOR operation to complement all 8 bits of the following byte:

```
        10011100
XOR     --------
        01100011
```

6. Give the 8 bit mask value and the logic operation to toggle bit #7 and bit #6 without disturbing any of the other bits.

7. If a value is ANDed with itself what would be the result?

8. If a value is ORed with itself what would be the result?

9. If a value is exclusive ORed with itself what would be the result?

10. IF a value is exclusive NORed (exclusive NOT ORed) with itself what would be the result?

11. Define the term toggle.

LAB #4
COMPUTER LANGUAGES

INTRODUCTION

In order for computers to be useful they must be told what to do. When a computer is told how to perform a given task it is said to be programmed. The process of programming a computer requires a computer language. A computer language, like a spoken language, provides a means for us to tell the computer what to do. You see computers do not yet speak fluent English. As with spoken languages, there are also many computer languages. Each computer language has its advantages and disadvantages depending on the type of program that is being written. All computer languages however, end up producing instructions that the particular computer understands. These instructions which the computer understands is referred to as the computers instruction set (the set of instructions understood by the computer). Not all computers have the same instruction set, in fact most computers have different instruction sets. A compiler or assembler, is used to translate the computer language statements of a computer program into the machine code (also referred to as object code) consisting of instructions from the computers instruction set. Compilers and assemblers are actually computer programs themselves, that are used to translate other computer programs. Once translated the computer can then execute the program.

Computer languages can be broken down into two categories, high level languages and low level languages. High level languages are more English like and much easier to program in then low level languages. Languages such as BASIC, PASCAL, and C are examples of high level languages. Low level languages are more cryptic and difficult to program in, but produce programs with less instructions, that run much faster then high level languages. Examples of low level languages are machine language and assembly language. Low level languages have a one to one correspondence with the computer's instructions. Each assembly language statement translates into one computer instruction, whereas with a high level language, one statement might translate into several hundred computer instructions.

When programming in machine language, the programmer acts as the translator. This is referred to as hand assembling because you are basically doing by hand what an assembler program does. The appropriate instruction is found, then the machine code is looked up and entered into the computer. A machine code is just a number that denotes a particular computer instruction specified by the manufacturer. On older computers the machine code was actually entered and displayed in binary using switches and lights, on computers of today this is rarely the case. Today, when machine code is used, it is usually entered and displayed in hex, with either the hardware or the software doing the conversion to or from binary. Remember computers at the lowest level (the machine level) must execute binary codes.

REFERENCE

The Intelligent Microcomputer by Goody, Chapter 13, starting at page 144, Chapter 14, page 160, and Chapter 8, pages 91 and 92.

OBJECTIVES

* Define computer programming.
* Define computer language.
* Understand the difference between low level and high level languages.
* Define machine language.
* Define assembly language.
* Know the five different 8085 instruction categories.
* Define op code.
* Define mnemonics.
* Understand flowcharting concepts.

PROCEDURE

The 8085 microprocessor has 246 instructions and each instruction is represented by an 8 bit binary value, which is called an op code or an instruction byte. The Instruction Set Encyclopedia ((c) Intel Corporation) included in appendix B, divides these instructions into five general categories which are as follows:

1) Data Transfer Group	Moves data between registers or between memory locations and registers.
2) Arithmetic Group	Adds, subtracts, increments or decrements data in registers or memory.
3) Logic Group	AND's, OR's, XOR's, compares, rotates or complements data in registers or between memory and a register.
4) Branch Group	Initiates conditional or unconditional jumps, calls, returns, and restarts.
5) Stack, I/O, and Control Group	Includes instructions for maintaining the stack, reading from input ports, writing to output ports, setting and reading interrupt masks, and changing the flag register.

Some machine language instructions require a byte or even two bytes of additional information often referred to as operands. The microprocessor reads the instruction and determines whether it requires an extra byte or two bytes. If the instruction requires another byte the 8085 will get the byte from the next consecutive address following the instruction byte, and if the instruction requires two bytes the 8085 will get them from the next two consecutive addresses following the instruction byte. With instructions that require two extra bytes, the byte following the op code is the least significant byte and the second byte after the op code is the most significant byte. To help you remember this order, remember that the high order byte is in the higher memory address than the low order byte.

Assembly Language

Intel (c) has designed a way to represent the 8085's machine language instructions using words called "mnemonics". Using these mnemonics, a language called "assembly language" was created which allows you to write machine language programs in a more readable form. Assembly Language programs are usally placed in a file known as the Source file. This file is read in by the assembler and converted to machine language. Two special files are created by the assembler the Object file and the Listing file. The Object file basically contains the machine language produced by the assembler. The Listing file is a text file that is a combination of the Source and Object files containing both the original assembly language with the corresponding machine language. Assembly language programs cannot be understood by the 8085 microprocessor until it is translated to machine language with a program called an "assembler". In the following lessons, programs will be listed in assembly language followed by the machine language translation of the program.

Example 4.0 lists an arbitrary assembly language program, showing the four basic fields of a line of assembly language: label, mnemonic, operand and comment. Note that assembly language programs usually don't have line numbers, but these are included to aid in explaining assembly language.

EXAMPLE 4.0

```
   LABEL    MNEMONIC    OPERAND        COMMENT
1  dips     equ         41h          ; port address for the dip switches
2  leds     equ         40h          ; port address of leds
3
4           org         8f01h        ; starting address of the program
5  loop:    in          dips         ; load A register with the dip
6                                    ; switch values
7           cma                      ; compliment A register to convert to
8                                    ; positive logic
9           out         leds         ; output the dipswitch contents to leds
10          cpi         00000000b    ; compare dipswitch to all zeros
```

```
11          jnz         loop        ; if dipswitch equals all zeros stop
12                                  ; else loop and try again
13          rst         7           ; stop program and return to MOS
14          end                     ; end of assembly language program
```

In order to make assembly language more readable and easier to modify, the provision was included which allows the use of a string of characters called a symbol or label to represent a numerical value. In the program above, the string of characters "dips" represents the value 41 hex, so line 7 which says "in dips" means, load the A register with the data from input port 41 hex (which is the dip switch port). As you can see, this is more readable than its assembly language equivalent "in 41h". The value of a symbol can be assigned using the EQU directive. Directives are used to instruct the assembler and are not part of the 8085 instruction set, nor are they ever executed by the program being assembled. Directives do not produce any machine code. The symbol on the left of the EQU directive is assigned the value on the right. As you can see in line 1 "dips" is assigned the value 41 hex and in line 2 "leds" is assigned the value 40 hex. Most assemblers assume that a number is decimal unless it is otherwise noted. Binary numbers are indicated by ending with a "b" or "B". Hex numbers must begin with a decimal number and end with "h" or "H". If they don't begin with a decimal number, the assembler will think they are labels, as in the case of the hex numbers DEAFh or BADh (they look like words instead of numbers). Start the hex numbers with 0 to solve this problem (0DEAFh, 0BADh).

The ORG directive tells the assembler the starting address in memory of the instructions that follow it. Line 4 shows that the program is to be assembled starting at address 8f01h.

When a symbol is in the label field of a line, and there is a mnemonic and not the directive EQU for that line, its value will be made the memory address of the opcode represented by the mnemonic. In line 5of the program above, "loop" is assigned the value of the memory address containing the opcode of the "in" mnemonic. Line 11uses the value of the label "loop" to produce the machine language for the "jnz" instruction.

Every character after a semicolon is considered a comment. Comments are used to describe the workings of a program to a person who might be reading the assembly language. They have no effect on the program because when the assembler encounters the ';' it ignores the rest of the characters on the current line.

A directive not included in the above program is the DS (Define Storage) directive. The DS directive tells the assembler to set aside the number of bytes of memory specified by the value to the right of the mnemonic. This memory is reserved for the storage of data instead of machine language. The locations allocated using the DS directive will generally change (like variables) as the program executes and accesses these locations. These locations must be initialized by the program at run time.

Another directive not included in the above program is the DB (Define Byte) directive. The assembler directive DB takes the data that follows the mnemonic and stores the value(s) of the data in memory. The values can be binary, hex or decimal numbers or a mixture of these but each number always uses one byte of memory. If the number is too large to be stored in one byte of memory the assembler will give an error message. A DW (Define Word) directive is provided for 16 bit data. The locations allocated by the DB or DW directives generally will not change (like constants) as the program executes and accesses these locations. These locations are usually initialized at assemble time.

The directive END tells the assembler that this is the end of the program. The directives END, EQU, ORG, DB and DS are all called "pseudo-ops", which means they are not translated into machine language directly. These pseudo-ops are used by the assembler to aid in assembling the source program (the original assembly language program). Lines 5 through 13contain mnemonics that actually correlate directly to machine language.

Below is the corresponding machine code in hex if the above program was assembled.

EXAMPLE 4.1

```
8F01    DB
8F02    41
8F03    2F
8F04    D3
8F05    40
8F06    FE
8FF7    00
```

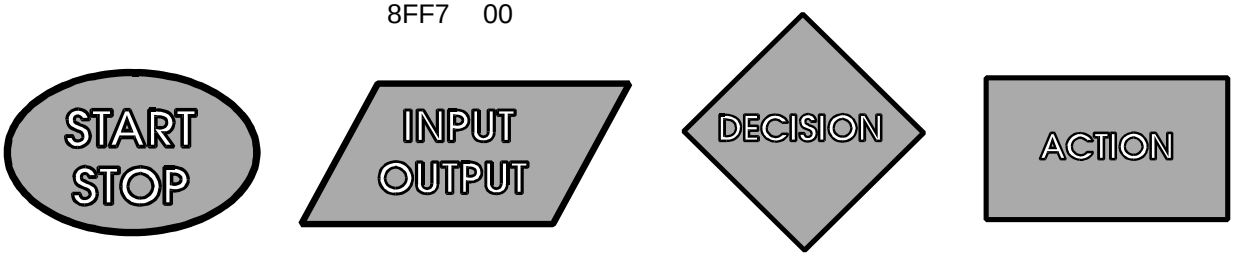


FIG. 4.0

```
8F08    C2
8F09    01

8F0A    8F
8F0B    FF
```

This code could now be entered into the trainer and executed.

Flowcharting

Before a program is actually written, a flowchart is usually constructed detailing the program algorithm (solution to the programming problem). A flowchart is a graphical representation of the operation of the program. A flowchart is composed of
several different types of graphical blocks connected with lines showing program flow. The four most important of these blocks is shown in Figure 4.0.

An oval shaped block is used to indicate the start or the end of a flowchart segment. A parallelogram is used to indicate input or output, for instance reading from the DIP switches or writing to the LEDs. A diamond shape block is used when making a decision, such as is the value greater than the setpoint yes or no. A rectangular block is used to indicate action, for example turn light on, sound alarm, etc. There are many other specialized symbols, but they are not required within this lab book.

When programming in assembly language, particularly for programs of 50 or more lines (statements/instructions), flowcharting and good documentation is essential. Assembly language programs are inherently difficult to understand, especially if someone else wrote the program. Flowcharts and good commenting practices go a long way in making a program easily readable and maintainable.

Flowcharts are not only used to aid in making a program more understandable, but also in the development of the program itself. A programmer typically will start with a high level flowchart depicting the operation of the program. With a high level flow chart, each block of the flowchart can represent hundreds of assembly language statements. The high level flowchart is an overview of the system. Each block of the high level flowchart is then broken down into a intermediate or low level flowchart. With a low level flowchart, each block may translate into one to five assembly language instructions. From the low level flowchart, the programmer writes the program. This high to low level process is known as top-down development. For simple programs a low level flowchart may be all that is necessary or for very simple programs no flowchart at all may be necessary.

Listed below is a high level BASIC program to count down from 10 and to 0 and exit, and the same program written in assembly language and machine language. Following these program examples is the corresponding low level flowchart. Note the flowchart is the same for all three examples.

BASIC PROGRAM EXAMPLE 4.2

```
LINE #.     STATEMENT                 COMMENT
1           LET COUNT=10              SET COUNT TO 10
2           LET COUNT=COUNT-1         DECREMENT COUNT BY 1
3           IF COUNT<>0 THEN GOTO 2   GOTO LINE #2 IF COUNT IS NOT 0
4           STOP                      STOP PROGRAM EXECUTION
```

ASSEMBLY LANGUAGE EXAMPLE 4.3

```
LABEL       OPCODE      OPERAND       COMMENT
            ORG         8F01H         START PROGRAM AT ADDRESS 8F01 HEX
START:      MVI         A,10          SET REGISTER A TO 10
LOOP:       DCR         A             DECREMENT A BY 1
            JNZ         LOOP          JUMP TO LOOP IF A IS NOT 0
            HLT                       HALT PROGRAM EXECUTION
```

MACHINE LANGUAGE EXAMPLE 4.4

```
ADDRESS     CONTENTS    INSTRUCTION   COMMENT
8F01        3E          MVI  A,10     SET REGISTER A TO 10
8F02        0A                        SECOND BYTE OF INSTRUCTION
8F03        3D          DCR  A        DECREMENT A BY 1
8F04        C2          JNZ  8F03     JUMP TO 8F03 IF A IS NOT 0
8F05        03                        LOW ORDER ADDRESS
8F06        8F                        HIGH ORDER ADDRESS
8F07        76          HLT           HALT PROGRAM EXECUTION
```

The low level flowcharting example below for simplicity sake does not contain any input or output. As you can see the flowchart's graphical representation eases program understanding.

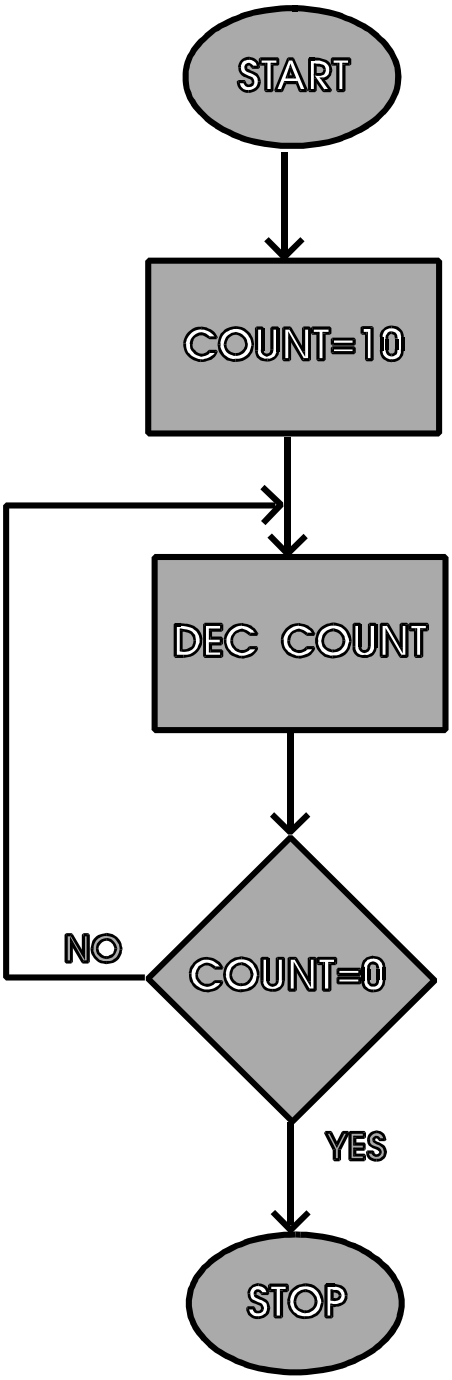


FIG. 4.1

This program example is very simple in nature, allowing an almost one-to-one correspondence between BASIC and Assembly language, in most cases this is not so. The simplicity of the example program requires only a low level flowchart to completely represent the program.

EXERCISE 4.0

1. Hand assemble the following mnemonic instructions into machine code. A table in appendix B contains each instruction with its corresbonding op code. Some instructions are 1, 2, or 3 bytes in length depending on the operands of the instruction.

```
                ORG         8F01
8F01  _______   RAR
8F02  _______   MOV         A,B
8F03  _______   ADD         C
8F04  _______   MVI         D,0A3H
8F05  _______
8F06  _______   LHLD        1234H
8F07  _______
8F08  _______
8F09  _______   ANI         101B
8F0A  _______
8F0B  _______   RST         7
```

QUESTIONS 4.0

1. What is the importance of a computer language?

2. Why would a program be written in assembler instead of BASIC?

3. Why would a program be written in C instead of assembler?

4. What language is programmed using strictly numbers?

5. When entering machine code into memory why is it better if the code is in hexadecimal form?

6. Each computer (machine) instruction is referenced by a numeric code called an _____.

7. What is the minimum number of bytes of a 8085 computer instruction?

8. What is the maximum number of bytes of a 8085 computer instruction?

9. The first byte of a computer instruction is always the _____.

10. Subsequent bytes of a computer instruction are always the _____.

11. What is another name for assembler directives?

12. What is the purpose of assembler directives?

13. All comments in an assembly language program must begin with which character?

14. Symbolic addresses used in assembly language programs are referred to as _____.

15. In the machine code example 4.1, why is the first instruction at address 8F01.

17. In the machine code example 4.1, the contents of addresses 8F01 and 8F02 were translated from what line in the corresponding assembler program of example 4.0?

18. The file that contains the original assembly language program is called the _____ file.

19. The file created by the assembler which contains only the machine code is called the _____ file.

20. The file created by the assembler which contains both the assembly code and the machine code is called the _____ file.

21. Why is it important to construct a flowchart before starting to write a program?

22. Modify the flowchart of Fig. 4.1 in order to have the program count to 10 and repeat indefinitely.

LAB #5
USING THE MONITOR OPERATING SYSTEM

INTRODUCTION

A computer operating system (OS) is a computer program that oversees the operation of the computer. It is responsible for initializing the system and controls the entry and display of data. The OS also provides services, that user written software can take advantage of, such as displaying numbers. If there were no OS services, the programmer would have to write the software necessary to perform these tasks.

A monitor assists the programmer by allowing the viewing and alteration of memory and the ability to run programs. Other monitor functions can include single stepping of programs, examining and altering the contents of CPU registers, and setting breakpoints. Combining the monitor and the operating system provides an ideal training environment, where programs can be entered, run and tested. The Monitor Operating System (MOS) provides all of the features mentioned above and more.

REFERENCE

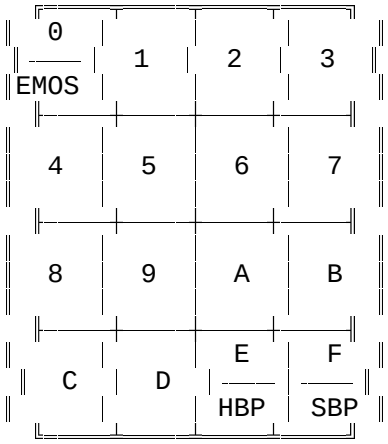
The Intelligent Microcomputer by Goody, Chapter 18, page 225, Monitor Programs.

OBJECTIVES

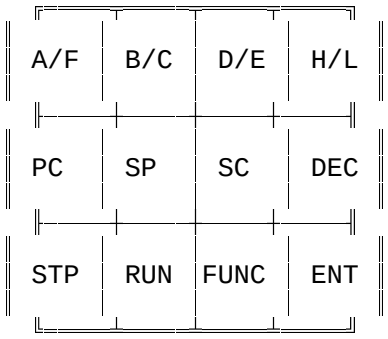
* Define operating system.
* Define monitor.
* View and change the contents of memory.
* View and change the register contents.
* Run a program.

PROCEDURE

The Monitor Operating System (MOS) is a powerful software program that provides the user with the tools to enter and edit programs as well as run, test, and debug machine language programs. MOS uses 8, fourteen segment alphanumeric LEDs for display output. This type of display is similar to the displays used in calculators, but allows letters in addition to numbers. The display at power-up reset, contains the default PC address (8F01H) in the left four characters of the display. This is what is referred to as the ADDRESS FIELD. In the right two characters are the contents of the displayed address. This field is known as the DATA/OP FIELD . The MOS uses a twenty-eight key keypad for input. Each key has tactile feel (clicks when pressed) and produces an audible tone when pressed providing feedback for the user. The keys are in two groups of sixteen and twelve. The group of sixteen are Hexadecimal numeric keys 0 - F, with 0, E and F having dual functions that will be discussed later.



The group of twelve keys are command keys. These keys allow the user to examine registers, change their contents, single step, and a variety of other functions.



KEYPAD DESCRIPTION

KEY(S)	DESCRIPTION
0-F	Numeric keys. When a numeric key (0-F HEX) is pressed, the numeric value appears at the right side of the data field display. To correct an entry error just repeat, using the correct number key and the error will be overwritten.
A/F	This key displays the contents of the A register (Accumulator) and the condition Flags. The A register and the flags are displayed as four HEX digits. The two digits on the right of the display represent the contents of the A register and the two on the left represent the condition Flags. The displayed value may be changed by entering the desired HEX number using the numeric keys and then pressing the ENT key. The condition Flags are defined bit by bit as follows:

BIT 0 CARRY FLAG (a set condition indicates a carry)

BIT 1 NOT USED

BIT 2 PARITY FLAG (a set condition indicates an even number of bits, odd parity)

BIT 3 NOT USED

BIT 4 AUX. CARRY FLAG (a set condition indicates a carry from bit 3 to bit 4)

BIT 5 NOT USED

BIT 6 ZERO FLAG (a set condition indicates a zero result)

BIT 7 SIGN FLAG (a set condition indicates bit 7 of the A register is a 1)

For example, if the display reads: 0044 A/F
This indicates the A register contains 0 and the ZERO and PARITY Flags are both set.

B/C This key displays the contents of the B/C register pair. The B/C register pair is displayed as four HEX digits. The two digits on the right of the display represent the contents of the B register and the two on the left represent the contents of the C register. The displayed value may be changed by entering the desired HEX number using the numeric keys and then pressing the ENT key.

D/E This key displays the contents of the D/E register pair. The D/E register pair is displayed as four HEX digits. The two digits on the right of the display represent the contents of the D register and the two on the left represent the contents of the E register. The displayed value may be changed by entering the desired HEX number using the numeric keys and then pressing the ENT key.

H/L This key displays the contents of the H/L register pair. The H/L register pair is displayed as four HEX digits. The two digits on the right of the display represent the contents of the H register and the two on the left represent the contents of the L register. The displayed value may be changed by entering the desired HEX number using the numeric keys and then pressing the ENT key.

PC This key displays the contents of the PROGRAM COUNTER. The PROGRAM COUNTER is a 16 bit register, displayed as four HEX digits. The displayed value may be changed by entering the desired HEX number using the numeric keys and then pressing the ENT key.

SP This key displays the contents of the STACK POINTER. The STACK POINTER is a 16 bit register, displayed as four HEX digits. The displayed value may be changed by entering the desired HEX number using the numeric keys and then pressing the ENT key.

SC This displays the two bytes that are at the top of the stack, or in other words, the data that would be removed from the stack if a POP instruction were executed. The left two displays show the byte at SP + 1 (stack pointer address + 1) and the two displays to the right of this represent the byte at SP. The displayed value may be changed by entering the desired hex number using the numeric keys and then pressing the ENT key.

DEC This key allows the user to decrement through memory, thus subtracting 1 from the PC every time the DEC key is pressed. The current PC address (4 HEX digits) and the contents of that address (2 HEX digits) are displayed. The address contents may be changed by entering the desired HEX number using the numeric keys and then pressing the ENT key.

STP This key causes the microprocessor to execute one instruction (single step) at the current PC address. Single Stepping is a valuable debugging tool that allows the user to examine registers and memory after each instruction executes. The STP key invokes a software single stepper that executes one user instruction and then returns to Monitor Operating System ready to except another user command. Unlike the Hardware Single Stepper, the Software Single Stepper requires the processor be active throughout the single stepping process. If single stepping a Service Call, the processor will execute the Service Call at full speed and stop at the instruction immediately following the Service Call.

RUN This key cause the microprocessor to execute a program at full speed starting at the current PC address. The program will continue to execute until an optional hardware or software breakpoint is encountered, a key is pressed, or until a RST 7 (0FFH) instruction is executed.

FUNC This key selects the second function of the keys that have two functions. When this key is pressed "FUNCTION" will appear on the displays and if a key is pressed which has a second function, that function will be performed.

ENT When the UT is turned on or reset the MOS is in data entry mode. When in this mode, pressing ENT will cause the data which is shown on the right two displays to be stored into the address pointed to by the PC register (which is shown on the left four displays) and the PC register will be incremented. When registers or hard or soft breakpoints are being displayed, pressing the ENT key causes the data that is being displayed to be stored in that register or for that breakpoint to be set. If in any mode you have typed some numbers and you want to restore the original value, you can restore them if you haven't press ENT yet. Press the "Func." key twice and you will be back in data entry mode and the data, registers or breakpoints will retain their original values.

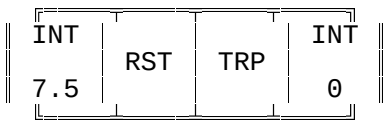
(Below are the second functions of the dual function keys)

EMOS This starts the EMOS (Extended Monitor Operating System). This requires that a PC or dumb terminal be connected to COM1.

HBP This key displays the current Hardware Breakpoint address. The displayed value may be changed by entering the desired breakpoint address in HEX using the numeric keys and then pressing the ENT key. The act of pressing the ENT key is what arms the Hardware Breakpoint arming circuit. A hardware breakpoint can only occur when the circuit is armed. Upon the occurrence of a Breakpoint, the Hardware Breakpoint address is still maintained, however the Hardware Breakpoint arming circuit is disarmed. To rearm the hardware breakpoint for the same address simply press the HRD BRK key followed by the ENT key. NOTE: If the program execution never accesses the breakpoint address, the program can not stop at the breakpoint address. The hardware breakpoint, breaks at any address including those in EPROM and operand addresses (see Section on breakpoints for additional information). This function is not provided on Revision 0 boards.

SBP This key displays the current Software Breakpoint address. If 0000 is displayed, then no breakpoint has been established. The displayed value may be changed by entering the desired breakpoint address in HEX using the numeric keys and then pressing the ENT key. Upon the occurrence of a Breakpoint the Software Breakpoint address is automatically reset to 0000. NOTE: If the program execution never reaches the breakpoint address or the breakpoint address is not that of an opcode, the program will not stop at the breakpoint address. Also any address specified that references addresses in EPROM (addresses below 8000 HEX) will not stop execution, even if the opcode at that address is executed (see section on breakpoints for additional information).

The UT has an additional 8 keys that are not included in the two groups mentioned above. Four of these are used to control the hardware single stepper circuit and will not be discussed at this time. The other four located near the power switch are as follows:



Pressing the INT 7.5 key will produce a high signal on the INT 7.5 line of the 8085 so you can manually induce an interrupt. The RST key, when pressed, will produce a low signal on the RESET IN* line, resetting the UT, changing the PC register to 8F01 and the SP register to FFD4, and clearing the other registers. The

memory contents, however, will stay intact. Pressing the TRP key will produce a high on the 8085's TRAP input causing control to return to MOS while retaining the values of the registers that were present when the key was pressed. The INT 0 key will produce a high signal on the 8259 interrupt controller's IR0 input line when it is pressed allowing you to manually induce an interrupt.

USING THE UNIVERSAL TRAINER

Before applying power to the UT, inspect all jumpers to determine if they are in their default positions. Table 5.0 below, lists each jumper and its default setting for board revisions 1&2. See drawing 2 for jumper locations.

Make any corrections that are necessary by moving the jumper from the incorrect position to the position indicated in Table 5.0. When moving jumpers, be sure power to the system is off. Once the jumpers are in their correct positions, no further adjustment is necessary until instructed to do so in later labs.

The UT is rugged but it can be damaged. Please handle it with reasonable care. The underside of the board is purposely exposed to allow hardware troubleshooting, be careful not to place it on any object that could cause damage.

The heatsinks on the linear regulators get hot. Avoid touching them.

There is no high voltage on this board, so the operator is safe from electric shock. The UT can not be damaged by the operator by moving jumpers or pressing buttons in any sequence. It is also impossible to damage the trainer through software provided there is no additional external circuitry connected to the UT.

There are DIP (dual inline package) switches on the UT that are required in several labs to be switched. Please do not use a lead pencil or ink pen for this purpose because the lead or ink can get into the switch.

DEFAULT JUMPER POSITIONS FOR BOARD REVISIONS 1&2 (TABLE 5.0)

JP1	**Position**	**A**
JP2	**Position**	**1000**
JP3	**Position**	**+5V**
JP4	**Position**	**16K**
JP5	**Position**	**8K**
JP6	**Position**	**I/O ONLY**
JP7	**Position**	**1-2**
JP8	**Position**	**Not Critical**
JP9	**Position**	**Not Critical**
JP10	**Position**	**Not Critical**
JP11	**Position**	**Not Critical**
JP12	**Position**	**Not Critical**
JP13	**Position**	**Not Critical**
JP14	**Position**	**Not Critical**
JP15	**Position**	**+V**
JP16	**Position**	**9600 (or to your terminal's rate)**
JP19	**Position**	**A**
JP20	**Position**	**B**
JP21	**Position**	**C**
JP22	**Position**	**A**
JP23	**Position**	**B**
JP24	**Position**	**A**
JP25	**Position**	**B**
JP26	**Position**	**A**

JP28	**Position**	**B**
JP29	**Position**	**2-3**
JP30	**Position**	**2-3**
JP31	**Position**	**A**
JP32	**Position**	**B**

To power-up the UT, first insert the wall transformers plug, in the power jack on the UT. Next plug the wall transformer into a standard 120VAC power socket and flip the on/off switch to the on position. The LED display shows 8F01on the left, two arbitrary digits on the right, and a tune is played through the speaker. This indicates that the power-up self test has been successfully completed and the monitor operating system is waiting for a command. If at any point in the following labs you press the wrong key, and you cannot recover from the mistake, simply press the reset button and the UT will reinitialize, allowing you to continue. If the reset button does not clear the problem, then turning the power switch off for several seconds and then back on will clear the problem.

In the following sections of the lab you will be introduced to the monitor operating system. Feel free to at any time refer back to the key descriptions until they become familiar to you. From this point on you will be working with the UT with power applied, remember the power-up sequence mentioned above when applying power to the UT.

VIEWING AND CHANGING MEMORY CONTENTS

When you first turn on the trainer, the "ADDRESS FIELD" display will show "8F01" and the "DATA/OP" display will show some random byte in hex. The number on the "ADDRESS FIELD" display is the value of the program counter register (PC) and the number on the "DATA/OP" display is the data at the memory address ***pointed to*** by the program counter. On power up the PC = 8F01 and "points" to the data at that memory address which is displayed in the DATA/OP field. If the data at that memory address happened to be the hex number DB then "DB" would be shown on the "DATA/OP" display. Below is an illustration of the program counter pointing to the data at memory address 8F01. The values shown in the memory addresses in the diagram most likely are not the actual values that are in the Trainer's memory. The memory contents are mostly random values when the Trainer is first turned on. The values below were chosen just as examples.

```
ADDRESS     CONTENTS
8F01        DB          <-PC
8F02        41
8F03        D3
:
: (memory addresses 8F04-FFFF)
```

When an address and the data at that address is displayed, the Trainer is in "data entry mode". Press the "enter" key and the PC register will be incremented to 8F02 and shown on the "ADDRESS FIELD" display and the data pointed to by the new value of the PC will be shown on the "DATA/OP" display. If the data at memory address 8F02 happened to be 41 hex, then "41" would be shown on the "DATA/OP" display. The program counter pointing to the data at memory address 8F02 is illustrated below.

```
ADDRESS     CONTENTS
8F01        DB
8F02        41          <-PC
8F03        D3
:
: (memory addresses 8F04-FFFF)
```

Press ENT several times. Note that the program counter incremented with each press. Press the DEC key and the value displayed for PC will be decremented and the number on the "DATA/OP" display is the byte of data pointed to by the new PC memory address. Press the DEC key until PC is 8F01 again or press the reset button.

Type "F" and "C", and you will see the hex number "FC" on the "DATA/OP" display. If you type a wrong digit, or you want to change the value you typed, just type the two correct hex digits and they will overwrite the current contents of the display. Now type "0" and "7" and press ENT and the PC will be incremented and the PC value 8F02 will be displayed along with the data pointed to by the new value of PC. Type the number 55 and press the "Dec." key. You will see that the data at address 8F01 is now 07. Press the "enter" key and you will see that the data at address 8F02 is not 55 as was typed before. This is because hex numbers that are typed aren't stored until the "enter" key is pressed. This is a useful feature when you have typed a number and decide you do not want it to be stored in memory or wish not to change the original value. Note, memory values in EPROM cannot be changed since EPROM like ROM is read only. In the above example the data is entered into RAM memory.

LOADING A PROGRAM INTO MEMORY

In the lessons that follow, it is necessary to load programs into memory. The machine language for the programs are listed in the same format as the following:

PROGRAM EXAMPLE 5.0

ADDRESS	CONTENTS	INSTRUCTION
8F01	DB	IN 41
8F02	41	
8F03	D3	OUT 40
8F04	40	
8F05	C3	JMP 8F01
8F06	01	
8F07	8F	

Before entering a program into memory, press the reset button. This resets the general purpose registers and flag register to zero and sets the stack pointer to FFD4 and the program counter to 8F01. Look at the program data list above. In order for a machine language program to be loaded into memory properly, the addresses under the column marked "ADDRESS" must contain the data to the right of them in the column marked "DATA" after you have completed loading the data. The column marked "instruction" just tells what instruction the data stands for, so this can be ignored. Since the PC value is 8F01, type DB, which is the data from the table that belongs in that address, and press "enter". The PC is now 8F01 so type 41 and press enter. Continue typing the data which belongs in the current PC address and pressing enter until all the addresses listed above have been loaded. Now press the DEC key and verify that the data at the current PC address is the same as the data in the program data table. If the data is not the same, just type the correct number and press enter and then continue pressing the DEC key and verifying data until the program counter is 8F01 again. Once you become comfortable with entering programs into memory you may want to skip the verification process. Don't be alarmed if you type a program incorrectly and it doesn't work, because it is impossible for a program to damage the Trainer. The worst thing that can happen is that you would have to press the reset button, or if the program was corrupted you may have to enter the program into memory again.

VIEWING AND CHANGING REGISTER CONTENTS

Examining the values of the 8085's registers is a valuable learning aid. To view the contents of a register you simply press the key corresponding to the register pair you want to examine. Note general purpose registers and the accumulator and flags can only be examined and modified in pairs.

Each line in the table below shows the key that should be pressed and the data that will be shown on the display as a result.

ADDRESS/REGISTER PAIR DISPLAYS

KEY	DATA DISPLAYED ON LEFT	DATA DISPLAYED ON RIGHT
A/F	= A register	Flag register
B/C	= B register	C register
D/E	= D register	E register
H/L	= H register	L register
TS	= Byte pointed to by SP+1	Byte pointed to by SP
SP	= High order byte of SP	Low order byte of SP
PC	= High order byte of PC	Low order byte of PC

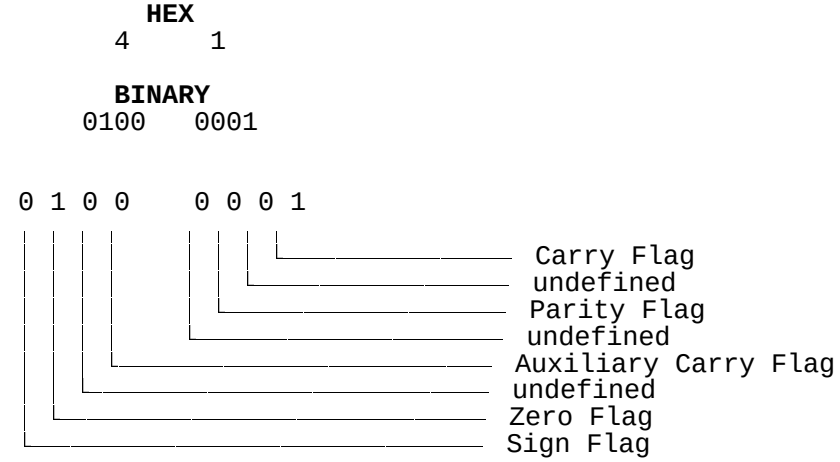
Pressing one of the above keys will cause the value in the registers to be shown on the two (4 digits) rightmost displays known as the "REGISTER FIELD" and the register name to be shown in the leftmost display. Pressing the DEC key will cause the Trainer to return to the data entry mode without any changes to registers or breakpoints.

If you want to change the value of the A register, press the "A/F" key and as a result the displays show "A/F 0000". To change the value of the A register to 1F hex without changing the flag register, press the following keys in order: "1","F","0","0", ENT. After pressing the "enter" key the display will no longer show the A register and flag register, instead it will return to the data entry mode. View the PSW (the A register and flag register) again by typing the "A/F" key. You will see that the values that were entered before are still in the registers. Notice that in order not to change the value of the flag register, it was necessary to enter its value again (00). If the flag register had been 40 and you wanted to change the A register to 2B without changing the flag register you would have to press the following keys (don't do this, this is just an example): "2", "B", "4", "0", ENT.

Now we will change the value of the flag register and we will assume that it doesn't matter what the value of the A register is. To make the flag register 41 hex type "4", "1". Notice that after pressing these two keys, the two zeros have been shifted to the A register display. Now press ENT, then view the A register and flag register again and you will see the new value (0041 hex) that was just entered.

As described earlier, the flag register is an 8 bit register which has individual bits called flags, that indicate the result of arithmetic or logical operations. To see what the values of the individual bits are, you must convert the hexadecimal value of the flag register to binary. If you don't remember how to do this, refer back to the lab on Computer Math in **LAB #2**. For example, to find out what the current flag values are, convert 41 hex to binary.

FIG. 5.3



According to the diagram above, the Carry and Zero flags have a value of 1 and the Sign, Auxiliary Carry, and Parity flags have a value of 0.

SOME TIPS ABOUT CHANGING REGISTERS

You can point the PC (program counter) register to any location by changing the PC register's contents. This eliminates the need of pressing the ENT or DEC keys many times in order to view or change data at memory addresses that are distant from the current address. If you are typing a new value for a register and you have not yet pressed the ENT key, you can exit to the data entry mode without changing the register by pressing the DEC key.

RUNNING A PROGRAM

Before running a program, it is very important that the PC be placed at the opcode of first instruction to be executed. The PC can placed at this address by using the DEC and ENT keys to position the PC without changing the program. An alternative method requires modifying the PC directly using the PC register key as explained above. Once the PC is correctly positioned at the opcode of the first instruction press the RUN key to execute the program. To stop the execution of a program which is running, simply press the TRP (trap) or RST (reset) buttons which are located in the top left corner of the board. When TRP is pressed, control is returned to MOS immediately and the registers will show the values that they had the moment TRP was pressed. Pressing RST will change the registers to the values they had after the board was first powered up.

Verify that the program entered in the LOADING A PROGRAM section (PROGRAM EXAMPLE 5.0) located at address 8F01 is still intact, if not reenter the program. Place the PC at address 8F01 the starting address of the program. To execute the program press the RUN key, the LED display will indicate that the program is running. Test the program by flipping the dipswitches, each switch affects its corresponding output LED. The program details are not important at this time and will be covered later in other labs.

EXERCISE 5.0

1. Enter the following hex bytes into the memory locations shown below and verify the memory contents for accuracy.

```
8F01  11
8F02  04
8F03  03
8F04  2E
8F05  06
8F06  63
8F07  24
8F08  3E
8F09  01
8F0A  47
8F0B  07
8F0C  4F
8F0D  97
8F0E  FF
```

2. The addresses 8F01 to 8F0E now contains the machine code of a short program. Run this program starting at address 8F01 (the first op code of the program) and examine the register contents after its execution. To run the program change the contents of the PC to 8F01, then press the RUN key. Fill in the register and flag contents below.

```
PC_____ B____ C____ D____ E____ H____ L____ A____ Sf__ Zf__ ACf__ Pf__ Cf__
```

3. When programming in machine language, the programmer translates each mnemonic instruction by hand, into machine code. This is referred to as hand assembling the program (see Lab #4). When programming in assembly language this translation is accomplished by the assembler. The translating back from machine code into mnemonic instructions is referred to as disassembling the

program. Disassemble by hand the program of exercise question 5.0.1 starting at address 8F01 and list the address and the mnemonic of each opcode below. The first address and mnemonic is given. A table in appendix B lists each instruction and its corresponding op code. There are 10 op codes (instructions) in the program. Remember that each instruction can have 1, 2, or 3 bytes depending on the operands.

```
1.    8F01  LXI   D,0304H
2.    _____ _________________
3.    _____ _________________
4.    _____ _________________
5.    _____ _________________
6.    _____ _________________
7.    _____ _________________
8.    _____ _________________
9.    _____ _________________
10.   _____ _________________
```

QUESTIONS 5.0

1. When the Trainer is powered up or reset what are the values of the registers and the flags?

PC_____ B_____ C_____ D_____ E_____ H_____ L_____ A_____ Sf___ Zf___ ACf___ Pf___ Cf___

2. MOS does the initialization of all the registers mentioned in question #1. Which of these registers is the only register actually initialized by the 8085 microprocessor prior to the MOS initialization and to what value?

3. When displaying the Accumulator and the Flags which register is displayed on the right two digits?

4. In order to execute a program that is in memory, what first must be done?

5. Why does the ENT/INC key have two names ENT and INC?

6. Memory locations below address 8000h cannot be changed, why?

7. What type of memory must be used to enter or change programs?

8. Why is it important for the Monitor Operating System to be in nonvolatile memory?

9. List a practical use for a disassembler.

10. What software is responsible for producing the audible beep when the Trainer is reset?

11. Explain the relationship between the dipswitch ON/OFF positions and the output ON/OFF status of the LEDs, when running the PROGRAM EXAMPLE 5.0.

12. Explain the function of the RST. key.

LAB #6
SOFTWARE SINGLE STEPPING AND BREAKPOINTS

INTRODUCTION

Being able to run a program at the fastest possible speed is important to maximize throughput. On a PC this translates to less time the user has to wait for a response. Sometimes during program development however, it's beneficial to slow the processor down so as to see what is going on internally. This process allows the user to examine the registers and memory locations after each instruction. If this process is done an instruction at a time, where the user has control after the execution of each instruction, then it is referred to as single stepping. If this process is performed on a number of instructions, where the user has control after the execution of that number of instructions, then it is referred to as tracing. A program being traced still displays the registers and possibly certain memory locations after each instruction executes. Single stepping and program tracing are important tools in debugging and developing programs.

There are two forms of single stepping, software single stepping and hardware single stepping. Software single stepping as the name implies is performed entirely in software. Embedded within MOS is the program that performs the single stepping function. This program executes one instruction and saves the contents of the registers, then returns to MOS allowing the user to enter another command. The user can then view the contents of the registers if need be. Hardware single stepping on the other hand is done entirely through hardware. A special circuit allows one instruction or portion of an instruction to execute and then asserts a low on the ready line providing a perpetual wait state. During this wait state the alphanumeric LEDs normally do not change as they are not receiving any instruction from the microprocessor. In order to see what the microprocessor is up to the discrete status LEDs are used. These LEDs are connected to the address, data, and control busses. By examining these LEDs and converting to Hex, the user can determine the address, the data, and the status of the data (memory or I/O, read/write). The hardware single stepper is useful for troubleshooting and understanding the hardware of the system.The software single stepper is most useful for development and debugging of software.

Another useful debug and development tool is the breakpoint. The breakpoint like the single step, comes in two varieties, software and hardware. The software breakpoint is accomplished through software by inserting a special software interrupt code in place of the instruction, at the address where you want to program to break. When this special instruction is executed, the program jumps back to MOS, which replaces the original instruction and returns control to the user. The breakpoint and single step functions when used together provide a powerful debugging aid. For example, if the user suspects a bug in the second half of a program, the user can set a breakpoint over the first half of the program, and begin single stepping at the suspect code. Hardware breakpoints are implemented in hardware and use comparitors connected directly to the address bus. When the address on the address bus matches the address in the comparitors the hardware single stepper is activated at the matching address. The hardware breakpoint can be used to arrive at the desired address to begin hardware single stepping.

REFERENCE

The Intelligent Microcomputer by Goody, Chapter 8.

OBJECTIVES

* To become familiar with the software single stepping process.
* To become familiar with the software breakpoint process.
* To become familiar with the following 8085 instructions.

IN <byte> opcode = DB

Load the A register with the data that is on the input port specified by the byte following the instruction. No flags are affected. This instruction is 2 bytes in length.

CMA opcode = 2F

The contents of the accumulator are complemented (zero bits become 1, one bits become 0; 1's complement or NOT function). This instruction is 1 byte in length.

OUT <byte> opcode = D3

Send the data in the A register to the output port specified by the byte following the instruction. No flags are affected. This instruction is 2 bytes in length.

JMP <addr> opcode = C3

Load the program counter (PC) with the address contained in the two bytes following the instruction. The first byte following the opcode is the least significant byte of the address, the second byte is the most significant byte of the address. No flags are affected. This instruction is 3 bytes in length.

PROCEDURE

Below is a flow chart of a simple program that reads the dipswitches (the 8 position dipswitch labeled S2/MANUAL DIGITAL INPUTS), complements the value, and outputs this value to the output port's LEDs. The value of the dipswitch requires complementing due to its use of negative logic. In fact it is very common for digital I/O ports to use negative logic, due to the fact that more current can be sinked than sourced with these type of devices. The digital output port LEDs for example require a low to turn on. A inverting buffer is used however to provide positive logic for this port. When using the dipswitches, rather than having to think in negative logic, complement the dipswitch value immediately after inputting the value. You can then treat the dipswitch value as a positive logic value.

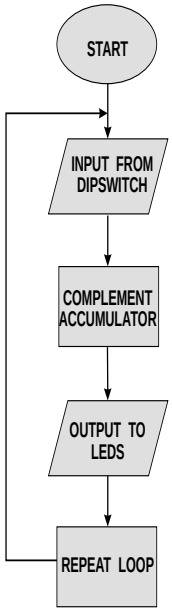


FIG. 6.0

There is another twist to reading the dipswitch and writing to the LEDs. Both the dipswitch and LEDs are referenced in a conventional control manner. Using conventional control I/O the number is read from left to right (note the numbering printed on the dipswitch). As programmers we think in binary and we also do all of our I/O in binary. It would be convenient, then, to think in binary terms where the numbers are read from right to left. Unfortunately most people do not think in binary so conventional control I/O is used and translated through the use of software. The translation process will be covered in a later lab.

ASSEMBLER PROGRAM EXAMPLE 6.0

```
leds        equ   40h
dips        equ   41h

            org   8F01h
loop:       in    dips                    ; load A register with
                                          ; dip switch values
            cma                           ; perform the NOT operation on the A
reg.
            out   leds                    ; output A register to LEDs
            jmp   loop                    ; jump to loop
            end
```

MACHINE LANGUAGE PROGRAM EXAMPLE 6.1

ADDRESS	**CONTENTS**	**INSTRUCTION**
8F01	DB	IN 41
8F02	41	I/O Address
8F03	2F	CMA
8F04	D3	OUT 40
8F05	40	I/O Address
8F06	C3	JMP 8F01
8F07	01	Low Address
8F08	8F	High Address

Notice in the machine language program EXAMPLE 6.1 that the JMP instruction occupies three memory locations. The first byte of the instruction is the opcode. The next two bytes of the instruction comprise the memory address where it will jump. Since all memory addresses in the 8085 are 16 bits and each memory location holds one byte, two memory locations are required to hold a memory address. When a 16 bit address or quantity is stored in memory, the low part of the address or number is stored in the lower address and the high part of the address or number is stored in the higher address. I/O addresses on the other hand, only require an 8 bit address, allowing the IN and OUT instructions to occupy just two bytes. The IN and OUT instructions are the only two 8085 instructions that allow writing to (OUT) or reading from (IN) an I/O device.

To run this machine language program, turn on the Trainer and the program counter address 8F01 will be shown on the left four displays. Load the machine language data from EXAMPLE 6.1 into memory. Once the whole program has been entered correctly return the program counter address to 8F01 by pressing the reset button, loading the PC register with 8F01, or using the DEC key. To run the program simply press the RUN key. The display will say RUNNING. You should observe that as you move the dip switches the eight digital output LED's (labled DIGITAL OUTPUT STATUS) will turn off or on accordingly. Since this program uses the JMP <address> instruction it will not stop until the RST (reset) button is pressed. Press the reset button and the program will stop, the digital output LEDs will turn off and the display will show "8F01 DB".

Single Stepping

The Universal Trainer allows you to execute a single machine language instruction starting at the address in the PC register; this procedure is called single stepping. After the instruction is completed, the PC register points to the address of the next instruction and the Trainer will be returned to the data entry mode. To single step the first instruction, press the STP key and the display will show "8F03 2F" which is the new

value for PC and the data pointed to by PC. The IN instruction was executed and the PC register was returned with the address of the next instruction, the CMA instruction. To single step the rest of the program, do the following steps:

1) Single step again and the display will show "8F04 D3". The CMA instruction has been executed and the PC register was returned with the address of the OUT 40 instruction.

2) Single step again and the display will show "8F06 C3". The OUT 40 instruction has been executed and the PC register was returned with the address of the JMP 8F01 instruction.

3) Single step again and the display will show "8F01 DB". This display shows 8F01 because that is the value that was loaded into the PC register by the JMP 8F01 instruction. You can see that the mnemonic came from the instructions ability to cause the program counter to "jump" to another location. PC now points to the IN 41 instruction again.

4) Single step again and the display will show "8F03 2F". The IN instruction has been executed and the PC register was returned with the address of the CMA instruction.

Right after step four, the complemented value of the dipswitch has been loaded into the A register and it can be viewed by pressing the "A/F" key. The value of the A register will be shown in the pair of digits to the left. Change the value of the A register to 0F Hex (Rememeber the A register is the pair of digits on the left. In this program the flag values don't matter so just enter 0F00.). Now do step one and then view the A Register. You will see the 1's complement of the value 0F. Do step two and you will see the digital output LEDs now show the complemented value of the A register. If you press STP four more times, the IN instruction will change the A register to the value of the dip switches again and the OUT instruction will display the new value of the A register on the digital output LEDs.

Breakpoints

Sometimes it is desirable to run full speed to a certain part of a program before single stepping, to run full speed through a long loop or subroutine, or to see if a program ever gets to a certain memory location. These things can all be accomplished through the use of breakpoints. The Universal Trainer features software breakpoints and hardware breakpoints. From a software standpoint we are primarily concerned with software breakpoints.

Setting a breakpoint on the Universal Trainer is similar to changing the contents of a register. Press the FUNC key then SBP and type the memory address of the breakpoint, then press the ENT key. The breakpoint is now set. When setting a breakpoint it is very important that the breakpoint memory address be that of an opcode. The program will not break if, the address of an operand is given, the opcode does not get executed, or the breakpoint address is in EPROM. If one of these breakpoint situations is desired the hardware breakpoint can be used. Once a program breaks the breakpoint is cleared and must be set again if another breakpoint is to occur.

To set a breakpoint in the above program we first select an address. For this procedure we will use the address "8F04" which contains the opcode "D3" of the OUT instruction. First press the SFT BRK key and enter the address 8F05, followed by the ENT key. Set the PC to address 8F06 and press the RUN key. The microprocessor will run the program at full speed breaking at address 8F04. At this point the user can examine registers, single step, or set a new breakpoint.

EXERCISE 6.0

Tracing a program as previously mentioned is a useful debugging aid. We can use the single stepper to trace a program by interrogating any registers or memory locations of interest, and logging this information

for each executed instruction. To trace the above program reset the Universal Trainer and verify the first line of the table below. Step the microprocessor and fill in the second line. Continue in this fashion until all lines of the table are complete. Before starting, put switches 1,2,3,6, and 7, in the ON position and 4, 5, and 8, are in the OFF position.

INSTRUCTION 0 PC 8F01 A 00 OPCODE DB M N E M O N I C I N 4 1

INSTRUCTION 1 PC _____ A _____ OPCODE _____ MNEMONIC _____

INSTRUCTION 2 PC _____ A _____ OPCODE _____ MNEMONIC _____

INSTRUCTION 3 PC _____ A _____ OPCODE _____ MNEMONIC _____

INSTRUCTION 4 PC _____ A _____ OPCODE _____ MNEMONIC _____

Before continuing, move DIP switches 5, 6, and 7, to the OFF position with all other switches ON.

INSTRUCTION 5 PC _____ A _____ OPCODE _____ MNEMONIC _____

INSTRUCTION 6 PC _____ A _____ OPCODE _____ MNEMONIC _____

INSTRUCTION 7 PC _____ A _____ OPCODE _____ MNEMONIC _____

INSTRUCTION 8 PC _____ A _____ OPCODE _____ MNEMONIC _____

Note that the opcode and mnemonic reflect the next instruction to execute. In this program the only registers that were utilized were the PC and the A register, requiring only these registers to be traced. Programs can be traced without the use of a microprocessor by determining how each instruction affects registers and memory and filling in an appropriate table by hand. This method of tracing a program is referred to as hand tracing. Hand tracing is often used when developing a program to verify the program execution.

QUESTIONS 6.0

1. As stated above all memory addresses of the 8085 microprocessor are 16 bits. What is the maximum amount of memory that the 8085 microprocessor can directly access?

2. As stated above, all I/O addresses of the 8085 microprocessor consist of 8 bits. How many different Read/Write I/O devices can the 8085 microprocessor directly access?

3. What type of single stepper utilizes MOS in order to provide single stepping?

4. What kind of breakpoint utilizes a special opcode inserted into memory to provide breakpoints?

5. If a software breakpoint was set at address 8F04 and the program was run starting at address 8F06, would the processor break?

6. If a software breakpoint was set at address 8F05 and the program was run starting at address 8F01, would the processor break?

7. Starting with the PC set at address 8F07, step 2 instructions and fill in the table below.

INSTR. 1 PC _____ A _____ OPCODE _____ MNEMONIC _____

INSTR. 2 PC _____ A _____ OPCODE _____ MNEMONIC _____

8. Explain the results obtained in question 6.0.7.

9. If the A register contained the value 5A Hex, what would be the value of the A register after executing the CMA instruction? What would be the value of the A register after executing two successive CMA instructions?

10. For each of the following instructions state YES or NO to whether the instruction affects any of the microprocessor flags (refer to Appendix B).

IN <BYTE> _____

OUT <BYTE> _____

CMA _____

JMP <ADDRESS> _____

11. After executing the following machine instruction, what address would be in the PC?

8F01 C3
8F02 50
8F03 05

12. When a software breakpoint occurs does the microprocessor stop running?

13. When a hardware breakpoint occurs does the microprocessor stop running?

14. Why would program execution not break if a software breakpoint was set to an address in EPROM?

LAB #7
LOADING REGISTERS

INTRODUCTION

Before a microprocessor can do any useful work registers need to be loaded with appropriate values. Some of these values may be 8 bit values and require storage in a single register. Other values may be 16 bits in length and require a register pair to hold the value. Values larger than 16 bits are generally stored in memory. Instructions that load registers, or move data from one place to another are referred to as Data Transfer Instructions.

REFERENCE

The Intelligent Microcomputer by Goody, Chapter 10.

OBJECTIVES

* To be able to understand the instruction coding process for the data transfer instructions presented in this lab.

* To be able to understand immediate addressing.

* To become familiar with the following 8085 instructions.

MVI r,<data8> opcode depends on operands

MVI r,<data8> is the move immediate instruction. This instruction is used to load binary 8 bit data (<data8>) into the various microprocessor registers. MVI r,<data8> is in the data transfer group of instructions and is a two byte instruction. The first byte of the instruction contains the opcode which tells the processor the register (r) where the data should be moved. The second byte contains the data. This instruction is 2 bytes in length.

LXI rp,<data16> opcode depends on operands

LXI rp,<data16> is the load register pair immediate instruction. This instruction is used to load binary 16 bit data (<data16>) into the various microprocessor register pairs. LXI rp,<data16> is in the data transfer group of instructions and is a three byte instruction. The first byte of the instruction contains the opcode, the next byte contains the low order data byte, and the last byte of the instruction contains the high order data byte. This instruction is 3 bytes in length.

RST 7 opcode = FF

The RST 7 is a special instruction that performs what is called a software interrupt. Whenever this instruction is executed control is transferred back to MOS, allowing the user to again enter commands. When writing a program on the Universal Trainer, this instruction is used to terminate (end) the program.

PROCEDURE

This lab focuses on the MVI r,<data8> and LXI r,<data16> data transfer instructions. The "I" in each of these mnemonics indicates that these instructions use immediate addressing. If an instruction uses immediate addressing the data is part of the instruction. In the case of the MVI r,<data8> instruction, the first byte of the instruction is of course the opcode and second byte of the instruction is the 8 bit data. The LXI rp,<data16> instruction is a three byte instruction, the opcode is followed by two bytes comprising the 16 bit data.

The immediate addressing mode is just one of four addressing modes (Immediate, Register, Direct, and Register Indirect) available on the 8085 microprocessor. The other three addressing modes will be presented in later labs. Different addressing modes allow data to be manipulated in different ways, and provides the programmer the ability to program efficiently. Some microprocessors such as the 8088 used in the PC, have several addressing modes not offered by the 8085. While these additional addressing modes make the 8088 processor more flexible, they also help contribute to its complexity when compared to the 8085 microprocessor.

Using The MVI r,<data8> Instruction

MVI r,<data8> is a two byte instruction, where the opcode is followed by eight bits of data. To code the MVI r,<data8> instruction, the first two most significant bits called the instruction code are 00. These two bits classify the class of instruction. The next three bits contain the register code that tells the microprocessor which register is the destination for the data. The registers are identified by their code in the following table:

TABLE 7.0 8 Bit Registers

Bits	Register Name
111	A
000	B
001	C
010	D
011	E
100	H
101	L

The final three bits are coded 110. These three bits are referred to as the instruction type and help further classify the instruction.

EXAMPLES 7.0

Here are two examples of how the syntax for an MVI r,<data8> instruction opcode is coded:

Example 1
To move data to register A (MVI A,<data8>), the opcode would look like this in binary form:

Class	A register code	Type
00	111	110

When converted to hexadecimal, the byte would look like this:

0011	1110
3	E

Example 2
To move data to the C register (MVI C,<data8>), the opcode would look like this in binary form:

Class	C register code	Type
00	001	110

Converted to Hexadecimal, the byte would look like this:

0000	1110
0	E

EXERCISE 7.0

To move the binary data 01011111 (5F) to the A register, you could use the following machine language program:

ADDRESS	**CONTENTS**	**INSTRUCTION**	
8F01	3E	MVI	A,5F
8F02	5F	Data	
8F03	FF	RST	7

MVI A,5F Moves the 8 bit data (5F) immediately following the opcode in memory (next address) to the A register .

RST 7 terminates the program.

Follow the steps below regarding the above program of EXERCISE 7.0.

1. Enter the above program into the trainer.

2. Now run the program by first returning the PC to address 8F01 and then pressing the RUN key.

3. After the program has run, Examine the A register by pressing the A/F key. The left two Hex digits display the data, 5F in the A register.

4. Change the above program so that the value placed into the A register is F5 rather then 5F.

5. Run the program from address 8F01 and verify your change.

6. Further modify the program so that F5 is now placed into the B register instead of the A register.

7. List the modified machine language program below.

ADDRESS	CONTENTS	INSTRUCTION
_____	_____	_____
_____	_____	_____
_____	_____	_____

Using The LXI rp<data16> Instruction

LXI rp,<data16> is a two byte instruction, where the opcode is followed by two bytes comprising a 16 bit data value. The first byte following the opcode is the low order byte of the data. The second byte following the opcode (third byte of the instruction) is the high order byte of the data. Remember for 16 bit values, the low order byte is in the lower memory address and the high order byte is in the higher memory address. To code the LXI rp,<data16> instruction, the first two most significant bits called the instruction code are 00. These two bits classify the instruction. Since this instruction is similar to the MVI instruction the class is the same. The next two bits contain the register pair code that tells the microprocessor which register pair is the destination for the data. The register pairs are identified by their code in the following table:

TABLE 7.1 16 Bit Registers

Bits	Register Pair Name
00	B --> IMPLIES B/C
01	D --> IMPLIES D/E
10	H --> IMPLIES H/L
11	SP

The "X" in the LXI mnemonic indicates that the instruction deals with 16 bit data values. Since certain instructions deal with 8 bit values and other instructions deal with 16 bit values, the operands for the mnemonics can be the same. For example the instruction MVI H,01 instruction would move the value 1 into the H register. Using the instruction LXI H,01 on the other hand would move the value 1 to the L register and the value 0 to the H register. Since the LXI instruction deals with 16 bit values, the value 01 is treated as a 16 bit value (think of it as 0001 in Hex) and as such the High order is loaded into the H register and the Low order is loaded into the L register. Note that 1, 01, or 0001 have all the same value of one. The MVI H,01 instruction and the LXI H,01 instruction use the same operand of "H", although for the LXI instruction H implies the H/L register pair.

The final four bits of the LXI instruction are coded 0001, which differentiates the LXI instruction from the MVI instruction. These four bits are used to help further classify the instruction.

EXAMPLES 7.1

Here are two examples of how the syntax for an LXI rp,<data16> instruction opcode is coded:

Example 1

To move data to register pair H/L (LXI H,<data16>), the opcode would look like this in binary form:

Class	H/L register code	Type
00	10	0001

Example 2

To move data to the Stack Pointer register (LXI SP,<data16>), the opcode would look like this in binary form:

Class	SP register code	Type
00	11	0001

EXERCISE 7.1

To move the 16 bit data value "1234" to both D/E and H/L register pairs, you could use the following machine language program:

ADDRESS	CONTENTS	INSTRUCTION
8F01	16	MVI D,12
8F02	12	Data
8F03	1E	MVI E,34
8F04	34	Data
8F05	21	LXI H,1234
8F06	34	Low Data
8F07	12	High Data
8F08	FF	RST 7

Follow the steps below regarding the above program of EXERCISE 7.1.

1. Enter the above program into the trainer.

2. Return the PC to address 8F01 and trace the program execution a step at a time (single step). Fill in the table below.

INSTR-PC	D	E	H	L	OPCODE	M N E M O N I C
0-8F01	00	00	00	00	16	MVI D,12
1-_____	____	____	____	____	____	_____
2-_____	____	____	____	____	____	_____
3-_____	____	____	____	____	____	_____

3. Change the above program so that the value 4321 is loaded into B/C register pair rather then loading 1234 into the H/L register pair..

4. Run the program from address 8F01 and verify your change.

5. List the memory values that are now present in the in addresses below.

   8F01 _____

   8F02 _____

   8F03 _____

EXERCISE 7.2

Programs are normally written in assembly language and then translated into machine language. This eases the burden of programming in machine language. Using instructions presented in this lab, translate (hand assemble) the assembly language program below into machine language.

```
one   equ   01
start equ   8F01h

      org   start

      lxi   h,6789h
      lxi   d,2345h
      mvi   a,one
      rst   7
      end
```

1. Verify the execution of the above program, by single stepping the resulting machine code.

2. Once verified list the machine language program below.

ADDRESS	CONTENTS	INSTRUCTION
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

QUESTIONS 7.0

1. How many bytes comprise the MVI instruction?

2. How many bytes comprise the LXI instruction?

3. How many bytes comprise the RST 7 instruction?.

4. The "I" in the MVI and LXI instructions indicates what?

5. The "X" in the LXI instruction indicates what?

6. Give the mnemonic for the following opcodes, by decoding each of the Hex values.

   36 Hex _____ 01 Hex _____

7. Define microprocessor addressing mode.

8. Define Immediate addressing.

9. Fill in the B and C register values after execution of the following machine code sequence.

```
8F01  01
8F02  11
8F03  00
8F04  03
8F05  22
8F06  FF
```

   B = _____ C = _____

LAB #8
TRANSFERRING DATA BETWEEN REGISTERS

INTRODUCTION

Once data has been loaded into a register, there are often times when a copy of the data needs to be placed in another register. Situations occur within programs that require the original data to be modified, making it necessary to use the copy that was placed in another register, for this purpose. More often than not however, the need to change registers is determined by the instructions being used. Certain instructions require the use of specific registers, facilitating the need to move information from one register to another. For example the CMA (Complement Accumulator) instruction operates on data held in the A register. The instructions covered in this lab belong to the Data Transfer group.

REFERENCE

The Intelligent Microcomputer by Goody, Chapter 8.

OBJECTIVES

* To be able to understand the instruction coding process of the MOV r1,r2 instruction.

* To be able to understand register addressing.

* To become familiar with the following instructions.

MOV r1,r2 opcode depends on operands

MOV r1,r2 is the move register instruction. This instruction enables the 8 bit contents of register r2 (source) to be copied to register r1 (destination). This instruction is 1 byte in length.

XCHG opcode = EB

Exchanges the 16 bit value in register pair H/L with the 16 bit value in register pair D/E. Register H is exchanged with register D and register L is exchanged with register E. This instruction is 1 byte in length.

PROCEDURE

This lab looks at two data transfer instructions that allow data to be moved or exchanged between registers. The MOV r1,r2 instruction uses register addressing to determine which registers are involved in the MOV instruction. With register addressing the instruction contains (in coded form) the registers used by the instruction. The MOV r1,r2 instruction can therefore be and is, a single byte instruction. The XCHG instruction is a fixed instruction, using the H/L register pair and the D/E register pair, so no operands are required. Since there can be no variation of this instruction no addressing mode is required. The XCHG instruction is also a single byte instruction.

Using The MOV r1,r2 Instruction

MOV r1,r2 is a single byte instruction, where the opcode is the only byte of the instruction. Using register addressing, no additional bytes are required. To code the MOV r1,r2 instruction, the first two most significant bits, called the instruction code, are 01. These two bits indicate the class of instruction. The next three bits contain the 8 bit register code that tells the microprocessor which register is the destination for the data. The last three bits of the opcode contains the 8 bit register code of the source register. The data is moved by the microprocessor from the source register to the destination register. The registers are identified by their code in the following table:

TABLE 8.0 **8 Bit Registers**

Bits	Register Name
111	A
000	B
001	C
010	D
011	E
100	H
101	L

EXAMPLES 8.0

Here are two examples of how the syntax for an MOV r1,r2 instruction opcode is coded:

Example 1
To move data to register A, from register B (MOV A,B), the opcode would look like this in binary form:

Instruction Class	Destination A register code	Source B register code
01	111	000

When converted to hexadecimal, the byte would look like this:

0111	1000
7	8

Example 2
To move data to the C register, from the D register (MOV C,D), the opcode would look like this in binary form:

Instruction Class	Destination C register code	Source D register code
01	001	010

Converted to Hexadecimal, the byte would look like this:

0100	1010
4	A

EXERCISE 8.0

1. Enter the program below into the trainer starting at address 8F01.

ADDRESS	CONTENTS	INSTRUCTION
8F01	01	LXI B,1234
8F02	34	Low Data
8F03	12	High Data
8F04	69	MOV L,C
8F05	60	MOV H,B
8F06	16	MVI D,56
8F07	56	Data
8F08	1E	MVI E,78
8F09	78	Data
8F0A	EB	XCHG
8F0B	FF	RST 7

2. Return the PC to address 8F01 and trace the program execution a step at a time (single step). Fill in the table below.

INSTR-PC	B	C	D	E	H	L	OPCODE	M N E M O N I C
0-8F01 00	00	00	00	00	00	01		LXI B,1234
1-_____	____	____	____	____	____	____	____	_____
2-_____	____	____	____	____	____	____	____	_____
3-_____	____	____	____	____	____	____	____	_____
4-_____	____	____	____	____	____	____	____	_____
5-_____	____	____	____	____	____	____	____	_____
6-_____	____	____	____	____	____	____	____	_____

3. In the above program of EXECISE 8.0, if instruction MOV L,C was changed to MOV L,B and instruction MOV H,B was changed to MOV H,C, what change could be made to instruction LXI B,1234 to allow the same final register contents as listed in INSTR. 6 above for registers D, E, H, and L?

4. Write the shortest machine language program possible in terms of bytes, to initialize the A, B, C, D, E, H, and L registers to zero. (Hint the program can be written in 8 or less bytes).

QUESTIONS 8.0

1. Give the mnemonic for the following opcodes, by decoding each of the Hex values.

   47 Hex _____ 6F Hex _____

2. If the program of EXERCISE 8.0 had the following modification, what would be the register contents after the RST 7 instruction executed?

MEMORY	CONTENTS
8F0B	EB
8F0C	FF

   Fill in the register contents in the blanks below.

   PC B C D E H L

   _____ ____ ____ ____ ____ ____ ____

3. Explain the results of question 8.0.2.

4. What happens if you move the contents of the A register to the A register? Is the instruction MOV A,A a valid instruction?

5. When the contents of the A register are moved to another register are the contents of the A register changed?

6. Give the Hex opcode for each of the following mnemonics.

   MOV H,L _____

   MOV B,C _____

7. In the program of EXERCISE 8.0, if a breakpoint is set at address 8F03 and run from address 8F01, does the processor break at address 8F03? Justify your answer.

8. Did any of the instructions in the program of EXERCISE 8.0 change any of the status flags? If so list the instructions that changed the flags?

LAB #9
INCREMENTING AND DECREMENTING REGISTERS

INTRODUCTION

The incrementing and decrementing of registers is a powerful feature. Using the increment instruction, we can count the occurrences of some event, like the number of items that have come down an assembly line or the number of people that have entered a building. Using the decrement instruction, we can set time limits on a particular activity or know how much of a commodity is left. Later we will see how the increment and decrement instructions can be used to sequence through memory to access data.

The increment and decrement instructions are part of the arithmetic group of instructions. The arithmetic group differs from the data transfer group in that no instructions within the data transfer group affected the flags. Most instructions within the arithmetic group however do affect the flags. Having the flags affected is important in making decisions (when time gets to zero, sound an alarm). This decision making process is what gives the microprocessor its "intelligence".

REFERENCE

The Intelligent Microcomputer by Goody, Chapter 11, starting at page 129.

OBJECTIVES

* To understand how the flags are affected by the increment and decrement instructions.

* To understand the coding process for the increment and decrement instructions.

* To become familiar with the following 8085 instructions

INR r opcode depends on the operand

The INR r instruction increments the 8 bit register r by the value of one. All flags except the carry flag are affected. This instruction is 1 byte in length.

DCR r opcode depends on the operand

The DCR r instruction decrements the 8 bit register r by the value of one. All flags except the carry flag are affected. This instruction is 1 byte in length.

INX rp opcode depends on the operand

The INX rp instruction decrements the 16 bit register pair rp by the value of one. No flags are affected. This instruction is 1 byte in length.

DCX rp opcode depends on the operand.

The DCX rp instruction decrements the 16 bit register pair rp by the value of one. No flags are affected. This instruction is 1 byte in length.

PROCEDURE

The increment and decrement instruction come in 8 and 16 bit versions of the instruction. As mentioned previously the "X" in the INX and DCX instructions indicate that these instructions deal with register pairs (16 bit values). Another difference between these instructions is the way they affect the status flags. The INR and DCR instructions affect all of the flags with the exception of the carry flag. The INX and DCX instructions do not affect any flags. These instructions belong to the arithmetic group and use register addressing to determine which register is incremented.

Coding the DCR r and INR r Instructions

The DCR r and INR r instructions are both single byte instructions and are coded similarly to the MVI instruction. The instruction code for both these instructions is 00. The next three bits is the register field, which tells the microprocessor which 8 bit register to increment or decrement. See TABLE 7.0, the table of 8 bit registers. The last three bits are the instruction type and is used to further classify the instruction.

EXAMPLES 9.0

Here are two examples of how the syntax for INR r and DCR r opcodes are coded:

Example 1
To increment register A (INR A), the opcode would look like this in binary form:

Class	A register code	Type
00	111	100

When converted to hexadecimal, the byte would look like this:

0011	1100
3	C

Example 2
To Decrement register D (DCR D), the opcode would look like this in binary form:

Class	D register code	Type
00	010	101

Converted to Hexadecimal, the byte would look like this:

0001	0101
1	5

Coding the DCX rp and INX rp Instructions

The DCX rp and INX rp instructions are both single byte instructions and are coded similarly to the LXI instruction. The instruction code for both these instructions is 00. The next two bits are the register pair field, which tells the microprocessor which 16 bit register pair to increment or decrement. See TABLE 7.1, the table of 16 bit register pairs. The last four bits are the instruction type and are used to further classify the instruction.

EXAMPLES 9.1

Here are two examples of how the syntax for INX rp and DCX rp opcodes are coded:

Example 1
To increment register pair B/C (INX B), the opcode would look like this in binary form:

Class	B/C register code	Type
00	00	0011

When converted to hexadecimal, the byte would look like this:

0000	0011
0	3

Example 2
To decrement the H/L register (DCX H), the opcode would look like this in binary form:

Class	H/L register code	Type
00	10	1011

Converted to Hexadecimal, the byte would look like this:

0011	1011
2	B

EXERCISE 9.0

1. Enter the program below into the trainer starting at address 8F01.

ADDRESS	CONTENTS	INSTRUCTION
8F01	01	LXI B,00FE
8F02	FE	Low Data
8F03	00	High Data
8F04	3E	MVI A,2
8F05	02	Data
8F06	16	MVI D,0FF
8F07	FF	Data
8F08	03	INX B
8F09	3D	DCR A
8F0A	14	INR D
8F0B	03	INX B
8F0C	3D	DCR A
8F0D	14	INR D
8F0E	16	MVI D,00
8F0F	00	Data
8F10	03	INX B
8F11	3D	DCR A
8F12	FF	RST 7

2. Return the PC to address 8F01 and trace the program execution a step at a time (single step). Fill in the register contents and flag (Zero and Carry) contents in the spaces provided below.

INSTR-PC	A	B	C	D	Zf	Cf	OPCODE	M N E M O N I C
0-8F01	00	00	00	00	0	0	01	LXI B,00FE
1-_____	____	____	____	____	____	____	____	_____
2-_____	____	____	____	____	____	____	____	_____
3-_____	____	____	____	____	____	____	____	_____
4-_____	____	____	____	____	____	____	____	_____
5-_____	____	____	____	____	____	____	____	_____
6-_____	____	____	____	____	____	____	____	_____
7-_____	____	____	____	____	____	____	____	_____
8-_____	____	____	____	____	____	____	____	_____
9-_____	____	____	____	____	____	____	____	_____
10-_____	____	____	____	____	____	____	____	_____
11-_____	____	____	____	____	____	____	____	_____
12-_____	____	____	____	____	____	____	____	_____

3. Change the LXI B,00FE instruction to LXI B,0FEFF. Set a breakpoint at address 8F0C and run the program starting at address 8F01. Examine the registers and flags after the break and fill in the blanks below.

PC	A	B	C	D	Zf	Cf	OPCODE	MNEMONIC
_____	____	____	____	____	____	____	____	_____

4. Replace all the INR D instructions with INX D instructions in the above program. Run the program starting at address 8F01 with no breakpoints. Examine the registers and flags after the program executes and fill in the blanks below.

PC	A	B	C	D	Zf	Cf	OPCODE	MNEMONIC
_____	____	____	____	____	____	____	____	_____

QUESTIONS 9.0

1. If the A register was initialized to zero and then incremented (INR A) 512 times, what would be the contents of the A register?

2. If the A register was initialized to zero and then decremented (DCR A) 512 times, what would be the contents of the A register?

3. If the B/C register pair was initialized to zero and then decremented (DCX B) 512 times, what would be the contents of the C register? What would be the contents of the B register?

4. Which of the instructions in the program of EXERCISE 9.0 affected the carry flag?

5. Which of the instructions in the program of EXERCISE 9.0 affected the zero flag?

6. Construct a two instruction program that uses the MVI instruction and a decrement instruction, to set the Zero flag to one.

7. In question 9.0.6, what value would be present in the A register when the Zero flag becomes one.

8. If the A register contained zero and the Zero flag was one, what increment instruction would clear the Zero flag to zero.

9. If the B register contained zero and the instruction MOV A,B was executed what flags would be affected.

10. Decode the following mnemonics to machine code.

    INX SP _____

    DCR L _____

LAB #10
PROGRAM LOOPING

INTRODUCTION

Frequently in programs it is desirable to execute a section of code a certain number of times. Rather than duplicate this section of code over and over, a program loop can be implemented. In LAB #6 we used a loop to continually check and output the dipswitch contents to the output LED's. The program of LAB #6 had no means of exiting the loop. A loop with no exit is referred to as a infinite loop, because it will execute the same instructions forever. A conditional loop, on the other hand, executes the instructions within the loop until the condition is met and then exits the loop. The branch group of instructions allow both conditional and unconditional (infinite) looping.

In order to implement a conditional jump, the flag register must be utilized. Certain instructions within the arithmetic group and logic group of instructions affect the flags. These instructions can then be utilized to make decisions on whether the program should loop or fall through (exit) the loop. This lab introduces all of the conditional jump instructions, but focuses on the jump instructions that reference the Zero flag.

REFERENCE

The Intelligent Microcomputer by Goody, Chapter 14.

OBJECTIVES

* To understand the concept of an unconditional jump.
* To understand the concept of a conditional jump.
* To be able to construct a delay loop.
* To be able to construct a program from a flowchart.
* To be able to exit a loop based on a comparison.
* To become familiar with the following instructions.

NOP opcode = 00

A special instruction that does no operation. The NOP instruction does not affect any registers or flags. This instruction basically does nothing but occupy time and one byte of memory space. This instruction is 1 byte in length.

Jccc <address> opcode depends on condition ccc

Jccc is the conditional jump instruction. ccc can be one of eight conditions referencing four of the status flags. If the selected condition is true, then program execution branches to the jump address <address>. If the condition is false then execution continues at the next instruction following the jump instruction. This instruction is 3 bytes in length.

CPI <data8> opcode = FE

Compare Immediate instruction. This instruction compares the byte following the opcode with the A register (Subtracts byte2 of the insrtuction from the A register) without changing the contents of the A register. This instruction only affects the flags. The carry flag = 1 if A is less than <data8> else the carry flag = 0. The zero flag = 1 if A equals <data8> else the zero flag = 0. All flags (Z, S, P, CY, AC) are affected by this instruction. This instruction is 2 bytes in length.

PROCEDURE

In this lab we analyze programs that utilize both the conditional and unconditional jump instructions. Refer back to lab #6 if necessary in order to refresh your memory about using the JMP instruction. It is often useful in programs to delay for a period of time before continuing. For example, a delay could be used in a burglar alarm to allow a person to shut off the alarm after they have entered the house. The NOP instruction can be placed in a delay loop in order to lengthen the delay, without affecting any flags or registers.

Using The Conditional Jump Instruction Jccc <address>

The conditional jump like the unconditional jump are 3 byte instructions. The first byte is the opcode, followed by the low order address, and then the high order address. Branch type instructions like the jump instruction have their own address modes,referred to as branch addressing modes. The 8085 microprocessor supports only one branch addressing mode called absolute addressing. Absolute addressing means the branch instruction contains the complete address of the jump. With absolute addressing we can jump anywhere we want within the 64K of memory space.

To code the Jccc <address> instruction, the first two most significant bits called the instruction code are both ones. These two bits classify the class of instruction. The next three bits contain the 8 bit condition code that tells the microprocessor on what condition to jump. The conditions are identified by their code in the following table:

TABLE 10.0 Conditions

ccc	Condition
000	NZ not zero (Zf = 0)
001	Z zero (Zf = 1)
010	NC not carry (Cf = 0)
011	C carry (Cf = 1)
100	PO parity odd (Pf = 0)
101	PE parity even (Pf = 1)
110	P plus (Sf = 0)
111	M minus (Sf = 1)

The final three bits are coded 010. These bits are referred to as the instruction type and help further classify the instruction.

EXAMPLES 10.0

Here are two examples of how the syntax for Jccc opcode is coded:

Example 1
To Jump Zero (JZ <address>), the opcode of the instruction would look like this in binary form:

Class	Z condition code	Type
11	001	010

When converted to hexadecimal, the byte would look like this:

1100	1010
C	A

Example 2
To Jump Not Carry, the opcode of the instruction would look like this in binary form:

Class	NC condition code	Type
11	010	010

Converted to Hexadecimal, the byte would look like this:

1101	0010
D	2

Although the decoding of all the conditional jump instructions were presented, this lab will emphasize the JNZ and JZ instructions. When using conditional jump instructions it important to realize that the flags are the basis of whether or not the jump takes place. This provides a very useful function in that the condition flags remember the results of an operation, allowing other instructions that do not modify the flags to execute before the conditional jump instruction is encountered. If an instruction sets a particular flag and it is the basis of a decision (conditional jump), no other instruction that modifies that particular flag should execute before the conditional jump, or the value of the flag may be changed and cause an incorrect jump.

As you can see from table 10.0, JNZ will occur when the result of an operation is Not Zero (Z=0) and JZ will occur when the result of an operation is Zero (Z=1). The zero flag may seem a bit confusing because when it is 1, it indicates an operation had a 0 result and when it is 0 it indicates a non-zero result. Just remember that a 1 indicates "yes" and 0 indicates "no". So, for example, when a decrement operation results in zero this causes the zero flag to be set to 1, signaling yes, the operation result was zero. If the operation did not result in zero, the zero flag would be 0 indicating no, the result was not zero.

EXERCISE 10.0

1. Enter the following delay loop program into memory starting at address 8F01.

ADDRESS	CONTENTS	INSTRUCTION
8F01	00	NOP
8F02	3E	MVI A,02
8F03	02	Data
8F04	00	NOP
8F05	00	NOP
8F06	3D	DCR A
8F07	C2	JNZ 8F04
8F08	04	Address low
8F09	8F	Address high
8F0A	3C	INR A
8F0B	FF	RST 7

2. Return the PC to address 8F01 and trace the program execution a step at a time (single step). Fill in the register contents and Zero flag contents in the blanks below.

INSTR-PC	**A**	**Zf**	**OPCODE**	**MNEMONIC**
0-8F01 00	0	00		NOP
1-_____	____	____	____	_____
2-_____	____	____	____	_____
3-_____	____	____	____	_____
4-_____	____	____	____	_____
5-_____	____	____	____	_____
6-_____	____	____	____	_____
7-_____	____	____	____	_____
8-_____	____	____	____	_____
9-_____	____	____	____	_____
10-_____	____	____	____	_____
11_____	____	____	____	_____

3. In the program of EXERCISE 10.0, how many iterations of the loop (times through the loop/number of jumps) did the program execute.

4. If in the program of EXERCISE 10.0, the MVI A,02 instruction was changed MVI A,00 how many iterations of the loop would be executed?

5. If in the program of EXERCISE 10.0, the JNZ 8F04 instruction was changed to JNZ 8F02 how many iterations of the loop would take place?

6. If in the program of EXERCISE 10.0, the JNZ 8F04 instruction was changed to JZ 8F04 how many iterations of the loop would be executed?

EXERCISE 10.1

In the program of EXERCISE 10.0, the delay is very short. The delay can be made longer by increasing the value of the A register. However even with the maximum delay set, the microprocessor can execute the program almost instantly. In order to get longer delays we can take advantage of a double loop. Enter the following double loop program into memory.

ADDRESS	CONTENTS	INSTRUCTION
8F01	01	LXI B,8000
8F02	00	Data low
8F03	80	Data high
8F04	0D	DCR C
8F05	79	MOV A,C
8F06	00	NOP
8F07	00	NOP
8F08	C2	JNZ 8F04
8F09	04	Address low
8F0A	8F	Address high
8F0B	05	DCR B
8F0C	C2	JNZ 8F04
8F0D	04	Address low
8F0E	8F	Address high
8F0F	00	NOP
8F10	FF	RST 7

1. Run the above program starting at address 8F01 with no breakpoints set. After the program finishes running fill in the register and flag contents below.

PC	A	B	C	Zf	Cf	OPCODE	MNEMONIC
_____	____	____	____	____	____	____	_____

2. Change the two NOP's at address 8F06 and 8F07 to an OUT 40 (two byte) instruction. Run the program full speed and determine how many times the eight output LED's are all on at the same time. (Hint, do not try and count the number of times the LED's are all on but try to calculate the value by determining the number of iterations of the loops).

3. Construct a flowchart for the modified program of EXERCISE 10.1.2 .

4. Change the contents of memory location 8F03 from 80 to 00. Does the program now take a longer time or a shorter time to execute?

5. Write an assembly language program and the corresponding machine code, that performs the actions of the flowchart below.

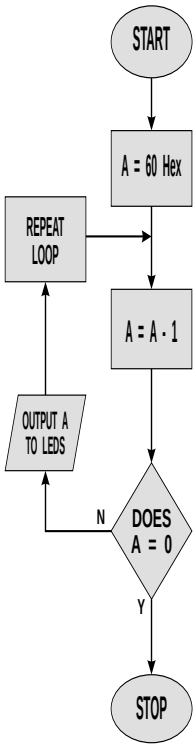


FIG. 10.0

The program of EXERCISE 10.0 utilized a down counter loop in order to achieve a short delay. The CPI <data8> instruction can be used to implement an up counter loop delay. Instead of using the DCR A instruction to set the Zero flag when the A register is empty, the CPI <data8> instruction will be used to set the Zero flag when the contents of the A register equals the value <data8> (second byte of the instruction). Figure 10.1 graphically describes how this program operates.

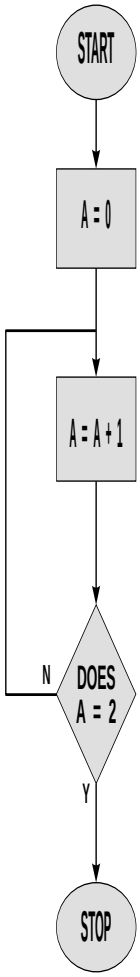


FIG. 10.1

EXERCISE 10.2

The assembly language program for the flowchart of figure 10.1 is listed below:

```
        org   8F01H
        mvi   a,0
loop:   inr   a
        cpi   2
        jnz   loop
        rst   7
```

1. For the assembly language program of EXERCISE 10.2 enter the corresponding machine language in the space provided below.

ADDRESS	CONTENTS	INSTRUCTION
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

2. Enter the machine language program of EXERCISE 10.2.1 into the trainer starting at address 8F01.

3. Return the PC to address 8F01 and trace the program execution a step at a time (single step). Fill in the register contents and Zero flag contents in the blanks below.

INSTR.	PC	A	Zf	OPCODE	MNEMONIC
0	8F01	00	0	3E	MVI A,0
1	_____	____	____	____	_____
2	_____	____	____	____	_____
3	_____	____	____	____	_____
4	_____	____	____	____	_____
5	_____	____	____	____	_____
6	_____	____	____	____	_____
7	_____	____	____	____	_____

4. Modify the assembly language program given in EXERCISE 10.2 so that the program counts from 10 to 20 (10 interations of the loop). List the modified assembly language program below.

5. Construct a flowchart and provide the resulting assembly language program to loop indefinitely until all of the dipswitches are in the ON position, whereby the loop exits and the output leds are subsequently all lit, and the program ends. (Hint: Refer to EXAMPLE 4.0).

QUESTIONS 10.0

1. A loop that uses an unconditional jump and has no exit is referred to as an infinite loop; Why is this so?

2. In order for the JZ instruction to jump the Zero flag must be at what state?

3. What flags, if any, affect the JMP instruction?

4. In the program of EXERCISE 10.1, if the DCR B instruction was replaced with the DCX B instruction, how many times would the DCX B instruction get executed?

5. In the program of EXERCISE 10.1, if a breakpoint was set at address 8F0B what value would be in the A register when the break occurred? Assume the program starting address is 8F01.

6. In the program of EXERCISE 10.0, if a DCR A instruction is placed at address 8F04 in place of the NOP instruction, how many iterations of the loop will take place?

7. In the program of EXERCISE 10.0, if a INR A instruction is placed at address 8F04 in place of the NOP instruction, how many iterations of the loop will take place?

8. If we wanted to exit a program loop when all the dipswitches are in the on position, explain how this could be done using the IN, INR A, DCR A, and JNZ instructions. (Do not use the CPI <data8> instruction).

LAB #11
HARDWARE TROUBLESHOOTING
USING THE MULTIMETER AND LOGIC PROBE

INTRODUCTION

This lab assumes you are using the B&K model DP-21 logic probe, and 2802B multimeter, supplied with the Universal Deluxe Package. Other brands and models may be used, but of course the detailed descriptions will probably not be exactly as stated here. Some labs require using a logic probe to read non-TTL level signals, so check the input voltage ratings of your probe before measuring these, since they may damage some probes.

There are many diagnostic tools used in finding hardware faults in computer systems. Two of these tools, the Multimeter and the Logic Probe, can be found on any technicians bench. With just the use of these two tools a number of hardware faults can be found and detected. These tools along with oscilloscope form the backbone of a technician's diagnostic toolbox.

The Multimeter can be used to measure a number of different electronic signals (multiple meters). Each of the different signals reveals a different piece of information about the electronic circuit being measured. The DC V (Direct Current Volts) setting is used to verify the DC voltage being applied to a circuit is correct. DC voltage is used to power almost all electronic computer circuits. Batteries provide DC voltage.

The AC V (Alternating Current) setting is used to measure AC voltages. A common source for AC voltages is the wall outlets in our homes. These outlets carry about 115 Volts AC. Computer circuits as mentioned earlier require DC voltage. But most computers plug into an outlet which carries AC voltage, which is one reason why computers require Power Supplies. A Power Supply converts the AC voltage from a wall outlet to DC voltages compatible with the electronic circuitry. The Power Supply used by the UT is a wall mount unit (the black square object that plugs into the outlet).

The OHMs (Ω) setting is used to measure the resistance or continuity of an electronic circuit. A Short circuit can be found in a electronic system by measuring zero or near zero OHMS between two points that would normally measure a higher resistance. The continuity of a wire or printed circuit board trace can be checked by measuring the resistance from end to end. If the resistance is high, then the wire or trace has an Open condition (wire has a break in it). A wire or trace that is good will read zero OHMS. If a good wire was placed across the terminals of a power supply or battery a short circuit would take place. Multimeters typically have an audible beeper used when checking continuity. This allows the technician to concentrate on the circuit and not have to keep watching the meter. The audible beep can be heard when a low resistance is measured.

The Logic Probe is a special digital electronic device that allows the technician to see the logical status of a test point in a circuit. The status of the point is indicated by multicolored LEDs, either high (RED), low (GREEN), open (LEDs are off), or pulsing (YELLOW). The rate of a pulsing signal can be determined by the rate at which the LEDs flash. By watching the LEDs the technician can easily see the digital status of a circuit point (high, low, open, or pulsing). The multimeter on the other hand provides analog readings (ie. 1.5VDC), where the logic probe just indicates levels.

When troubleshooting a circuit, chip reference numbers are given (ie. U12) and chip manufacturer's numbers (ie. 74LS132) may also be given in order to locate a test point. The reference number is painted on the circuit board and manufacturer's number is painted on the IC (Integrated Circuit) itself. Pin 1 of a standard DIP (Dual Inline Package) IC is sometimes marked with a dot. If no dot is provided a notch is cut out in the middle top of the IC. The first pin to the **LEFT** of this notch, with the notch pointing up, is pin 1. Numbering proceeds sequentially from pin 1 down the **LEFT** side of the IC, and then across the bottom and up the **RIGHT** side towards the notch at the top. The last pin on the IC is across from pin 1.

OBJECTIVES

* To become familiar with common electronic hardware terms.
* To understand the basic operation of the multimeter.
* To understand the basic operation of the logic probe.
* To understand basic troubleshooting concepts.

PROCEDURE

Turn the trainer off and connect the multimeter's (the shorter of the instruments with the dial) BLACK lead to the trainer's common ground, with the binding post labeled SCOPE/METER GROUND. This post has a hole in the side, permitting the post to be screwed down onto the back multimeter lead.

When troubleshooting microcomputers it is often necessary to discover shorts, opens, or verify continuity. The multimeter can provide this information when the function switch is moved to the OHMs setting. Usually before applying power to a newly manufactured computer, it is checked to make sure there are no shorts between Vcc (+5 VDC) and ground. To do this make sure the power to the board is off, then measure the resistance between Vcc (Place meter tip on TP6) and the ground terminal (Black lead labeled SCOPE/METER GROUND). On the Universal Trainer and you should see that it is more than 700 ohms. **NOTE**: If the resistance read on the meter has the letter **K** after it, this indicates the resistance is multiplied by **1000**.

The DC V function is used to measure voltage levels throughout the board being tested. Turn the dial on the multimeter to DC V. With this function enabled and the trainer on, you should measure approximately 5 Volts with the meter tip at TP6. The MC145406 RS-232 voltage generator chip (U57, the 16 pin chip close to COM1) has ≈ +8 volts on pin 1 and ≈ -8 volts on pin 8. Measure these voltages for yourself, they will be referred to later in the lab.

The Logic Probe

Connect the logic probe's power leads to the trainer's LOGIC PROBE POWER terminals, TP5 and TP6. Connect the RED clip to TP6 (+5 volt power), and the BLACK clip to TP5 (ground). Set the TTL / CMOS selector switch on the logic probe to TTL, and the MEM / PULSE selector switch to MEM. In this setting the logic probe is used to tell whether a signal is logic high or logic low or whether it has made a transition from high to low or vice-versa. Placing the Logic Probe on TP6 (+5V), will indicate a high and TP5 (ground) will indicate a low.

To see how the MEM function works, turn dipswitch 1 of the manual digital inputs to the **off** position then turn the MEM / PULSE switch to the PULSE position on the probe. Place the probe on pin 1 of the digital I/O connector above the dipswitch. Now move the MEM / PULSE switch to the MEM position. The yellow LED might light-up as soon as you do this, so to reset it, keep the probe on the pin and move the switch from MEM to PULSE and back to MEM be careful not to remove the probe from the pin. When you move dipswitch 1 to the **ON** position you will see the logic level change from high to low and the yellow LED will indicate the transition. You may also verify that the probe can indicate low to high transitions by performing the same procedure only starting with the dipswitch in its current position. The MEM function is useful to detect a signal that may occur only once during power-up of the board, such as a reset signal or a chip select during initialization of an I/O device. Once a signal has been detected, the probe must be reset before detecting the next signal.

Set the MEM / PULSE selector switch to PULSE. The logic probe can now be used to tell whether a signal is logic high or logic low or whether it is a pulsing signal. When examining pin 37 (fourth pin down on the right side of the chip) of U12 (clock-out of the 8085 microprocessor) the probe will indicate a pulsing signal by flashing the yellow LED. To better understand how the logic probe responds to different kinds of pulsing signals, load and run the following program.

EXERCISE 11.0

1. Enter the program below into the UT starting at address 8F01. This program will allow you to send a variable duty cycle square wave to the digital output lines (these may be accessed via the digital I/O connector). The duty cycle is controlled by dipswitch S2, the manual digital inputs.

ASSEMBLY LANGUAGE PROGRAM

```
adcin       equ   9           ; service to read A/D
portout     equ   40h         ; output port address
mos         equ   1000h
            org   8f01h
loop1:      mvi   d,0         ; counter
            mvi   a,0
            out   portout     ; output all lows
            mvi   e,0         ; choose A/D channel 0
            mvi   c,adcin
loop2:      call  mos
            mov   a,d         ; get count value
            cmp   l           ; see if D<L
            jc    skip        ; skip if D<L
            mvi   a,0ffh
            out   portout     ; output all highs when D>L
skip:       inr   d
            jnz   loop2       ; if not 0, read A/D again
            jmp   loop1       ; restart
```

MACHINE LANGUAGE PROGRAM

ADDRESS	DATA	INSTRUCTION
8F01	16	MVI D,0
8F02	00	
8F03	3E	MVI A,0
8F04	00	
8F05	D3	OUT 40
8F06	40	
8F07	1E	MVI E,0
8F08	00	
8F09	0E	MVI C,9
8F0A	09	
8F0B	CD	CALL 1000
8F0C	00	
8F0D	10	
8F0E	7A	MOV A,D
8F0F	BD	CMP L
8F10	DA	JC 8F17
8F11	17	
8F12	8F	
8F13	3E	MVI A,FF
8F14	FF	
8F15	D3	OUT 40
8F16	40	
8F17	14	INR D
8F18	C2	JNZ 8F0B
8F19	0B	
8F1A	8F	
8F1B	C3	JMP 8F01
8F1C	01	
8F1D	8F	

2. Run the above program, and probe any of the digital output lines of the digital I/O connector (even numbered pins 2-16). The smaller the binary value that is input from the dipswitches, the less the square wave will be at the high level during the period of the wave. The larger the value, the longer the square wave will be at the high level. The binary value that is input from the dipswitches may be determined by probing the odd numbered pins 1-15 of the digital I/O connector (a dipswitch in the ON position produces a binary 0 input, and in the OFF position produces a binary 1).

3. Probe the digital output lines and notice that the yellow light on the probe will be pulsing (if at least 1 dipswitch is on), and as the value input from the dipswitches approach either 0 or 255 only one of the logic indicators will appear to be lit, while the yellow light will continue pulsing. When it appears that only one logic indicator is **on**, this indicates that during the period of the signal, it is at that logic level the most often. Set the dipswitches to input 127 (01111111 binary) so that both logic levels appear to be on at the same time. Remember to input the value to the dipswitch backwards. This indicates a 50% duty cycle (approximately) square wave of less than 200KHZ. Both logic level indicators will be **off** when measuring a 50% duty cycle square wave greater than 200KHZ.

4. The multimeter although a valuable instrument, can give misleading information for certain types of signals (as most instruments can if used improperly). Put the multimeter on the DC V setting and measure one of the digital output lines again. As you change the DIP switches so that the value approaches 255 you will notice that the voltage measured will approach 5V, and conversely, as you change the DIP switches so the value approaches 0, the voltage measured will approach 0V. It gives no indication that there is a pulsing output, and at times, depending on the duty cycle, even seems to indicate that the voltages are not TTL level (normal TTL levels are 0 to 0.8V for a low and 3.5V to 5V for a high). When the voltage levels are changing rapidly,as in this example, the multimeter just displays the average of the voltage levels that are received.

5. The logic probe is also misleading at times. Remember the MC145406 (U57) RS-232 voltage generator? Use the logic probe to measure the levels present on pins 1 and 8 of this IC. You will notice that where you measured +8 volts earlier with the multimeter, the probe indicates a high and where you measured -8 volts the logic probe measures a low. There is no indication that the signal is negative or that it is not TTL level.

Remembering the relative strengths and weaknesses of the multimeter and logic probe will be helpful in the sections that follow.

TROUBLESHOOTING A FAILURE

We will now try to evaluate the effectiveness of the multimeter and the logic probe as a troubleshooting tools. As we have seen, each tool can reveal specific conditions present in a system under test, but sometimes the readings or results found by one tool are inconclusive, or illogical, but the other tool can reveal the problem to the technician. We will attempt to show the different characteristics and how to apply these tools in the following section.

EXERCISE 11.1

The first step will be to program a failure. Be sure the trainer is functioning properly before programming the failures, so you will have only one problem to contend with at a time. As mentioned previously, there are two types of Universal Trainers which are similar, though the program failure jumpering is more robust on boards of revision 1 or greater. This lab session will describe jumpering positions that will create the same problem on all revisions of the UT.

1. To program a failure turn off the trainer then move jumper JP21 (it is near the left edge of the board) from position C to position B (middle two pins jumpered). Power up the Trainer, and observe the response.

Observation: Is the trainer operating correctly ?
If not, where should I begin looking for the problem ?

Hint: There are at least two major areas of concern with a microprocessor-based system, foremost is the microprocessor "core", consisting of the microprocessor chip itself, some memory, and miscellaneous logic circuitry. But nothing in the system will operate if not powered correctly. The power supply of the system, although not very glamorous, is the very first thing that must be verified when dealing with an inoperative microprocessor system, or any other electronic device for that matter.

2. Connect the logic probe's power leads to the trainer's LOGIC PROBE POWER terminals, TP5 and TP6. Clip the RED clip to TP6 (5 volt power), and the BLACK clip to TP5 (ground).

3. Connect the multimeter's BLACK lead to the trainer's common ground, with the binding post labeled SCOPE/METER GROUND. This post has a hole on the side, permitting the post to be screwed down onto the probe. This post will also accept Banana Plugs, or plain wires as well.

4. Set the TTL / CMOS selector switch on the logic probe to TTL, and the MEM / PULSE selector switch to PULSE. Set the multimeter's function switch to DC V (DC volts).

5. What is the significance of the logic probe and multimeter settings?

6. Probe the following places on the trainer with both the multimeter and the logic probe. Record the readings for both instruments.

PIN DESCRIPTION		MULTIMETER READING	LOGIC PROBE READING
A. U12 pin 40 (Vcc)		_____	_____
B. U12 pin 20 (Vss/GND)		_____	_____
C. U12 pin 37 (CLK OUT)		_____	_____
D. U12 pin 36 (RESET IN*)		_____	_____
E. U12 pin 35 (READY)		_____	_____
F. U12 pin 39 (HOLD)		_____	_____
G. Test points TP2,	A0	_____	_____
	A1	_____	_____
	A2	_____	_____

	A3	_____	_____
	A4	_____	_____
	A5	_____	_____
	A6	_____	_____
	A7	_____	_____
H. Test points TP4,	DB0	_____	_____
	DB1	_____	_____
	DB2	_____	_____
	DB3	_____	_____
	DB4	_____	_____
	DB5	_____	_____
	DB6	_____	_____
	DB7	_____	_____

Each point given above has some special significance to point out the relative strengths or weaknesses between the two kinds of test instrument.

7. For each of the blanks below fill in which instrument (Multimeter or Logic Probe) was most accurate at reading the value for each test point of EXERCISE 11.1.6?

A._____ B._____ C._____ D._____

E._____ F._____ G._____ H._____

TEST POINT DESCRIPTIONS

Following is a description of the significance of each of the test points specified and the signals you should expect to see.

A. U12 pin 40.

This pin is the Vcc pin of the microprocessor, U12. All logic chips must have power, this place was simply a handy place to access Vcc. It also serves to verify one of the most important inputs to the chip as well. You should obtain a voltage reading of 4.8 to 5.2 volts, the normal Vcc tolerance for a TTL logic power supply. (Although most of the chips are CMOS, TTL logic levels are compatible for all but the most critical applications.) The logic probe reading should be a logic HIGH. Verification of proper Vcc supply operation is intended as the reason for taking a measurement at this point.

B. U12 pin 20.

This pin is the ground pin of the microprocessor. The voltage reading here should be almost zero volts, but since the multimeter can read millivolt levels, the student may record the millivolt drops produced along the copper tracks of the board. Note that millivolt readings in logic circuits are actually ground for practical reasons. The logic probe should read a logic LOW. This pin is another very important input pin, to verify that the microprocessor chip has power supply ground.

C. U12 pin 37.

This is the system clock output pin of the microprocessor chip. The logic probe will indicate a pulse signal and the DC voltage readings at this point will vary from board to board but they should be approximately +2.5 volts. This is not the actual voltage on the pin, it is the average of the 0-5 volt square wave. The signal at this point should demonstrate that at least part of the microprocessor chip is functioning.

D. U12 pin 36.

This is the reset input to the microprocessor chip. The purpose of probing this point is to verify the input state of the reset to the chip. This is an active low input signal, and in a powered up system, this line should be high if the microprocessor is to operate. This is a "must be correct" signal, because the microprocessor cannot operate unless it is in the proper state. Probe this line and note the action as the reset button is pressed and released. This line should be low when the system is initially powered up, or when the reset button is pushed, but after a moment, the input should change to a logic high.

E. U12 pin 35.

This is the READY input to the microprocessor chip. It must be asserted logic HIGH, or else the microprocessor will wait until this lead is active. Unless the student has accidently activated the single stepping controls on the trainer, the readings obtained should indicate a steady logic high on both the logic probe and the voltmeter readings recorded. This is another of the "must be correct" inputs to the microprocessor chip.

F. U12 pin 39.

This pin is the HOLD REQUEST input to the microprocessor, when asserted, it will inform the chip that some other device wants control of the system. The microprocessor will finish the current bus cycle, and then it will release the busses by going tri-state on key outputs. As there are no other bus masters in the system (in this lab), this input must not be asserted. Logic probe reading should be a logic LOW, and the voltmeter readings should also be no more than a few millivolts. This input is also one of the "must be correct" microprocessor inputs.

G. TP2, A0-A7.

These are the lower 8 bits of the current memory or I/O address. In most cases, when a microcomputer is working correctly, these bits should be changing rapidly. The situation where these might not be changing is when the EPROM program is in a loop that is very short and doesn't access the stack or any other RAM or I/O. This is usually quite rare, so a steady signal on any of these test points could indicate a problem.

H. TP4, DB0-DB7.

The test points DB0-DB7 are the AD0-AD7 pins of the microprocessor. It is more rare for one of the AD0-AD7 pins to be at a continuous level than for A0-A7, because A0-A7 have only 1 function (supplying the

lower 8 bits of the I/O or memory address), whereas AD0-AD7 have 2 multiplexed functions. At times AD0-AD7 function as the bi-directional data bus, and at other times they are used to output low address data to be latched by U16 (providing A0-A7). A steady signal on any of DB0-DB7 will most likely indicate a problem.

FAULT DETECTION

Notice from the measured readings that test points DB2 (the AD2 pin of the microprocessor) and A2 were both continuously low. This should not be the case. With the microprocessor running normally the DB2 line and the A2 line should be changing rapidly.

Now that the defective signals have been identified, steps to determine their nature and how to correct them must be taken. A way to do this involves using an multimeter in the OHMS mode to check the lines for physical shorts (when there is continuity where there should not be continuity), and opens (breaks in the printed circuit board traces, or bad solder joints).

8. Turn off the trainer and set the multimeter to the OHMS (Ω) mode, and check for shorts to ground at A2 and DB2. This can be accomplished by connecting the black lead to ground post terminal labeled SCOPE/METER GROUND and check with meter tip test points TP2, A2 and TP4, DB2. Fill in the measured resistances below.

   A2. _____ DB2. _____

   You should notice that there is a high resistance between A2 and ground, and very low resistance between DB2 (Pin AD2 on the microprocessor) and ground. This low resistance measurement indicates that AD2 is shorted to ground.

9. With the power to the unit off, reposition the fault jumper to it's original position. Then apply power to the unit and verify correct operation.

FAULT SOLUTION

If A2 is not shorted to ground, why is it continuously low when the trainer is running? This may be answered by analyzing how the 8085 low order address is latched. The designers of the 8085 microprocessor chip, in order to reduce the number of pins on the chip, allowed for multiple functions for its AD0-AD7 pins. At the beginning of each machine cycle, these pins perform the address function. The low order memory address is placed on them, then microprocessor signal ALE (Address Latch Enable) is strobed to cause chip U16 to latch the data from AD0-AD7 to its outputs which are then referred to as A0-A7. After the low order address is latched, the AD0-AD7 pins are switched over to the data bus function. Since the AD2 pin is shorted to ground you can understand why the U16 latch will output a low for A2.

CONCLUSION

Failure Jumper JP21 (or FJ21 for REV0) is one of the many programmable failures that can be activated on the trainer. It is used to short the microprocessor's AD2 line to ground (in the case of rev 1 or greater, it has an additional position to short AD2 to signal A5, but this advanced problem is not dealt with in this lab).

In a real world situation, such as production testing, this failure, a stuck logic line, can happen when a PCB trace has a copper short to ground, Vcc, or any other signal or power supply rail. This could be due to bad etching, or a wire lead clipping may have been stuck on the board during soldering, or in some cases the output or input of a chip on the logic line in question is shorted internally. It can also be due to a bad output driver or gate input.

QUESTIONS 11.0

1. What test instrument used is capable of giving analog readings?

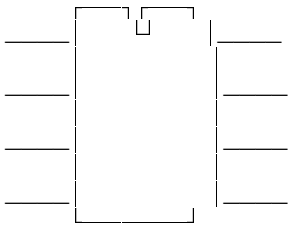
2. Which test instrument used is more proficient at detecting a rapidly changing signal?

3. Define continuity in terms of electronics.

4. What fault condition do you think could cause the most damage an open or a short?

5. What electronic device is responsible for changing AC to DC?

6. Label the pin numbers on the following IC below.



7. How can we tell the difference between a short, or a gate or buffer merely reflecting a short (a phantom problem) as was the case with signal line A2 in EXERCISE 11.1.

8. Which tool has proved most helpful in this lab?

9. Is there a trouble shooting phase where one tool is more valuable than the other?

10. What is the first thing to check when troubleshooting a piece of electronic equipment?

LAB #12
USING THE ENHANCED MONITOR OPERATING SYSTEM (EMOS)

INTRODUCTION

The Monitor Operating System (MOS) is a powerful software program that provides the user with the tools to enter and edit programs as well as run, test, and debug machine language programs. When the Universal Trainer is first turned on, interaction between the Monitor Operating System and the user is through the on-board keypad and display. This mode of operation is referred to as the Keypad/Display mode and was presented in LAB #5. The Keypad/Display mode requires no additional hardware and is therefore stand alone, making the UT quite portable.

The most productive way to use the Universal Trainer, however, is through EMOS in which all interaction is through a Terminal/PC. This mode obviously allows more information to be exchanged easier than with the keypad display mode. Additional functions/commands are available through the Terminal/PC mode that are not possible with the Keypad/Display mode.

OBJECTIVES

* Know how to invoke EMOS
* Know the advantages and disadvantages of using EMOS vs. MOS
* View and change the contents of memory using EMOS.
* View and change the register contents using EMOS.
* Fill and Dump memory using EMOS.
* Run a program using EMOS.

PROCEDURE

Before using EMOS, the serial cable must be connected from the UT to the Terminal/PC and the Terminal/PC's serial protocol must be set to 1 stop bit, 8 data bits and no parity. The UT's baud rate may be set by moving the communications baud rate jumper JP16 to the desired rate. The baud rate of the Terminal/PC should be set to match the UT. If a PC is being used, a terminal emulator software package such as ECOM (available from EMAC) must be used.

When the UT is properly connected to the Terminal/PC, invoke EMOS from MOS by pressing FUNC then the EMOS key. The LEDs will display **--EMOS--** , indicating that all communication with the UT will now take place through the COM1 RS232 serial port. The Terminal/PC should now display the following menu:

```
                    EMOS Vx.xx HELP MENU

B --> Bring Block from RAMDISK to Memory
C --> Change register contents
D --> Dump memory contents
E --> Edit memory contents
F --> Fill memory with byte
G --> Go execute program { full speed }
H --> Hex/Decimal math {1st + 2nd, 1st - 2nd}
I --> Input from I/O port
L --> List memory contents using mnemonics
M --> Move section of memory
O --> Output to I/O port
R --> display Register contents
S --> MOS Service call
T --> Trace program execution
W --> Write memory to RAMDISK
< --> hex download from trainer to host
> --> hex upload to trainer from host
? --> display this help menu
```

If the menu is not displayed type "?" and the menu should appear. The above Help Menu can be displayed at any time from the EMOS minus "-" prompt by typing a "?".

Each of the EMOS commands are invoked by typing a single character at the EMOS prompt. EMOS will prompt the user for any necessary command input and checks the input to be sure it is of the proper form. When an "ADDRESS" is requested, a number of up to 4 HEX digits should be entered. When a "BLOCK" is requested, a number of up to 3 DEC (decimal) digits should be entered. The <ESC> key or bad input will abort the issued command and cause a "?" to be displayed. A complete description of all the EMOS commands is given in the Universal Trainer Reference Manual. Several of these commands and their descriptions are covered in detail in this lab.

VIEWING AND CHANGING MEMORY CONTENTS

Using The Dump Command

The "D" command shows memory across the screen in Hex and ASCII both. ASCII stands for the American Standard Code for Information Interchange. The ASCII codes 00 through 7F in hex define control and printable characters. Each character has a unique ASCII code that references that character. For instance 31 Hex is the ASCII code for the character "1". If you send the value 31 Hex to most printers, they would print a one. Below is an example of an arbitrary Dump of memory.

```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
8F01 30 31 0D 0A 41 42 C3 FF 61 62 23 24 2A 20 00 3F  01  AB..ab#$*  ?
```

In this example 8F01 is the starting address of the Dump line. Sixteen memory locations starting at 8F01 through 8F10 are displayed in Hexidecimal form, followed by the same sixteen addresses displayed in ASCII form. ASCII control codes such as carriage return (0Dh) and line feed (0Ah) display a space character. Memory values above 7F display a period. All other values display their ASCII character representation. This standard dump format is featured on most microprocessor development systems. It is very useful for displaying a whole block of memory that may contain data, variables, or ASCII strings.

Steps for Dumping Memory

1. Type the letter "D" at the minus prompt.
2. Enter the desired starting address in Hex. If the address is less than four digits press the <ENTER> key.
3. Enter the number of bytes from 1 to FFF in Hex. A minimum of 16 bytes is always displayed. Pressing any key will pause the display. Pressing the <ESC> key at this time will abort the Dump command or any other key will continue.
4. After completion of the Dump command the minus prompt will be displayed, indicating EMOS is ready for another command.

Using The Edit Command

The "E" command displays memory a byte at a time with its associated address and allows the contents of the location to be optionally changed. Press the <ENTER> key to view the next sequential address' memory contents without changing the contents of the preceeding memory location. Typing a byte value in Hex (00 - FF) will replace the contents of the currently displayed address with the Hex value typed. Typing a minus sign followed by a single Hex digit (0 - F) will subtract the value of the digit from the current address, allowing you to back up, up to 15 memory locations. Pressing the <ESC> key at any time will return you to the

EMOS minus prompt. The Edit command is particularly useful for entering and editing programs.

Steps for Editing Memory

1. Type the letter "E" at the minus prompt.
2. Enter the desired starting address in Hex. If the address is less than four digits press the <ENTER> key after typing the digits.
3. Change the contents of each memory location as required. To change the memory contents, type up to two Hex digits.
4. After you finish Editing the memory contents, press the <ESC> key to return to the EMOS minus prompt.

Using The Fill Command

The "F" command is used to Fill a section of memory with a particular byte. This command comes in handy for determining what memory locations are being altered or accessed. This is done by filling the suspect memory section with a known byte. After executing a program the suspect memory section is Dumped to see which locations have changed. The Fill command requires a starting address, the number of bytes (memory locations) to Fill, and the byte with which to Fill the memory locations. This command will obviously have no effect when trying to Fill EPROM locations.

Steps For Filling Memory

1. Type the letter "F" at the minus prompt.
2. Enter the desired starting address in Hex. If the address is less than four digits press the <ENTER> key.
3. Enter the number of bytes from 1 to FFF in Hex. If the address is less than four digits press the <ENTER> key after typing the digits.
4. Enter the Fill byte (0 - FF).
5. After completion of the FILL command the minus prompt will be diplayed, indicating EMOS is ready for another command.

Using The List Command

The "L" command disassembles the memory contents, displaying the memory contents as 8085 mnemonics with their associated addresses. This command is very useful in verifying the instructions you entered using the Edit command. The List command requires a starting address. This address must be that of an opcode. If the address of an operand or data is given, the list command will assume this address contains an opcode and will disassemble it and the contents of subsequent memory locations, giving misleading results. The List command will disassemble sixteen 8085 instructions and will pause. Pressing the <ESC> key will return you to the minus prompt. Pressing any other key will continue the disassembling process. If an invalid (not defined) opcode is encountered, three question marks are displayed. The availability of the List command is one of the benifits to using EMOS in place of MOS.

Steps For Listing Memory

1. Type the letter "L" at the minus prompt.
2. Enter the desired starting address in Hex. If the address is less than four digits press the <ENTER> key after typing the digits.
3. Press any key to continue disassembling, or press the Escape key to return to the EMOS minus prompt.

LOADING A PROGRAM INTO MEMORY

In Lab #5 we loaded a program into memory using MOS. In this lab we will load the same program using EMOS. The program is as follows:

ADDRESS	DATA	INSTRUCTION
8F01	DB	IN 41
8F02	41	
8F03	D3	OUT 40
8F04	40	
8F05	C3	JMP 8F01
8F06	01	
8F07	8F	

Use the following steps to enter the above program using EMOS.

1. Use the "F" command to fill 16 memory locations, starting at address 8F01, with 0. This step is not mandatory in loading a program, but makes later verification of the program easier.
2. Type "E" to invoke the Edit command and specify a starting address of 8FO1.
3. At address 8FO1 enter the Hex value DB from the first line of the above program.
4. Continue entering the Data Instructions in their respective addresses.
5. Press the <ESC> key to return to the EMOS minus prompt.
6. Use the "L" command to list the contents of memory starting at memory address 8F01.
7. Verify that each memory location contains the correct value (compare instructions and operands). If a mistake is found repeat from step #1.

VIEWING AND CHANGING REGISTER CONTENTS

Viewing the register contents with MOS required that each register be viewed independently. EMOS offers the luxury of viewing not only all the registers, but the top of stack and the memory contents pointed to by the PC with its associated mnemonic, all at the same time. EMOS displays each flag of the flag register individually so the user doesn't need to decode a hex flag value into binary to see which flags are set or clear. EMOS is a more efficient and robust training environment than is MOS.

When using MOS to change register contents, each change affected a register pair. With EMOS each 8 bit register is changed and displayed individually and not as a register pair.

Using The Register Command

The "R" command displays the flags, all the registers, the contents of the top of stack, and the contents of memory pointed to by the PC with its associated mnemonic. Each flag is given an individual bit, making it easy to see individual flag status. All 8 bit registers are displayed individually and not as register pairs. The data in memory pointed to by the PC is displayed and referred to as an OPCODE. The memory contents is referred to as opcode because it is assumed that any location the PC points to should contain an opcode, even though it may contain an operand or data. The format for displaying the registers in EMOS is shown below.

SZxAxPxC	A	B	C	D	E	H	L	TS	SP	PC	OPCODE	MNEMONIC
00000000	00	00	00	00	00	00	00	0000	0000	0000	00	NOP

The flags read from the left are:	S	Sign flag
	Z	Zero flag
	x	Undefined
	A	Auxiliary Carry flag
	x	Undefined
	P	Parity flag
	x	Undefined
	C	Carry flag
The 8 bit registers from the left are:	A	Accumulator register
	B	B general purpose register
	C	C general purpose register
	D	D general purpose register
	E	E general purpose register
	H	H general purpose register
	L	L general purpose register
The 16 bit memory value:	TS	Top of Stack contents
The 16 bit registers from the left:	SP	Stack Pointer register
	PC	Program Counter register
The 8 bit memory value:	OPCODE	Value pointed to by PC
	MNEMONIC	Instruction Abbreviation

To display the registers, simply type the letter "R" at the EMOS minus prompt.

Using The Change Command

The "C" command allows the modification of the flags, registers, the top of stack, and the memory location pointed to by the PC. Each register can be individually changed with EMOS unlike with MOS. Flags, although individually displayed, must be changed as a Hex 8 bit value. The Change command first prints the current register contents, and then prompts the user with the following:

```
SELECT REG. [F, A, B, C, D, H, L, T, S, P, O]..
```

Select "F" to change the flags, "A" - "L" to change the 8 bit registers, "T" for the top of stack, "S" for the stack pointer, and "O" to change the memory contents pointed to by the PC. After entering one of the above letters, the user is then prompted for the new hex value for the selected register/memory location. Remember TS, SP, and PC all contain 16 bit values and all other selections contain 8 bit values. The Change command will display the register contents again for verification, after the change has been made and before returning to the EMOS minus prompt.

RUNNING A PROGRAM

To run a program using EMOS, the Go command is used. Using the Go command, the user can enter an optional starting address and/or optional breakpoint address. Then when the Go command prompts for the starting address and the <ENTER> key is pressed without typing a number, execution will start at the current PC value. The steps required to run a program are listed below.

1. Type the letter "G" to invoke the Go command.
2. Enter the starting address (in hex) of the instruction to be executed. If the address is less than four digits press the <ENTER> key after typing the digits.
3. At this time EMOS will prompt you for a breakpoint address. If a breakpoint address is not required, just press the <ENTER> key. (Breakpoints are not discussed in this lab).
4. The Terminal/PC will indicate that the program is running. To stop execution of the program and return to the EMOS minus prompt, press the trap button, (labeled TRP and located next to the RESET button).

Verify that the program entered in the LOADING A PROGRAM section, located at address 8F01 is still intact. If not, reenter the program. To execute the program use the Go command selecting the starting address of 8F01 with no breakpoint as mentioned above. The Terminal/PC will show the message "`RUNNING.. HIT TRAP BUTTON TO STOP`". Test the program by flipping the DIP switches and you should see that each switch affects its corresponding output LED.

When you press TRP, the program will stop, the registers will be displayed (just as if the "R" command had been executed) and the minus prompt will appear.

EXERCISE 12.0

1. Enter the following Hex bytes into the memory locations shown below and verify the memory contents for accuracy.

```
8F01  11
8F02  05
8F03  04
8F04  2E
8F05  07
8F06  63
8F07  24
8F08  3E
8F09  02
8F0A  47
8F0B  0F
8F0C  48
8F0D  0C
8F0E  FF
```

2. The addresses 8F01 to 8F0E now contain the machine code of a short program. Run this program starting at address 8F01 (the first opcode of the program) and examine the register contents after its execution. To run the program, use the Go command with a starting address of 8F01 and no breakpoint. After running the program, fill in the register and flag contents below.

```
PC_______ B___ C___ D___ E___ H___ L___ A___ Sf__ Zf__ ACf__ Pf__ Cf__
```

3. Using the List command, disassemble the program of exercise question #1 starting at 8F01. Fill in the mnemonics and operands below. The first mnemonic with operands is given. There are 10 opcodes (instructions) in the program and each instruction can have 1, 2, or 3 bytes depending on the operands.

```
1,    LXI   D,0405H
2.    ____________________
3.    ____________________
4.    ____________________
5.    ____________________
6.    ____________________
7.    ____________________
8.    ____________________
9.    ____________________
10.   ____________________
```

4. Using the "D" command, Dump 16 memory locations starting at address 8F01. Fill in the associated blanks below.

```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   012345678ABCDEF
8F01 __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __  ________________
```

5. Using the "F" command Fill 16 memory locations starting at address 8F01 with the ASCII character "+" (plus). The Hex value of "+" is 2B. To verify the Fill command, Dump 16 memory locations beginning at address 8F01 and fill in the associated blanks below.

```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   012345678ABCDEF
8F01 __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __  ________________
```

QUESTIONS 12.0

1. List two advantages and two disadvantages of using EMOS instead of MOS.

2. List the ASCII codes in hex for the string "8085 Universal Trainer" . (See ASCII table in Appendix A)

3.. What EMOS command could be used to verify that the ASCII string of question #2 is correct?

4. ASCII is an acronym for

5. What ASCII character is used as the EMOS prompt?

6. What ASCII character is entered to display the EMOS Help Menu?

7. What button or key is pressed to stop executing a program and return to the EMOS prompt?

8. What button or key is pressed to stop Editing and return to the EMOS prompt?

9. To back up 10 bytes when using the Edit command (without exiting), you would enter _____ .

10. If a data entry error is made using an EMOS command, what key would abort the command and return you to the EMOS prompt?

11. Are the registers initialized to the same values for EMOS as they are for MOS? If not, what register values are different?

12. If a program is entered in MOS can it be executed in EMOS? If a program is entered in EMOS can it be executed in MOS?

13. If a register is changed in MOS does it remain changed in EMOS?

APPENDIX A

I/O Device Map:

Device	I/O address(s)	Memory address(s)
Hardware Breakpoint	**00 to 02 hex**	**6000 to 6002 hex**
Low Address	00	6000
High Address	01	6001
Arming Circuit	02	6002
8259 Interrupt controller	**10 to 11 hex**	**6010 to 6011 hex**
8253 Timer	**20 to 23 hex**	**6020 to 6023 hex**
Timer #0 (prescaler)	20	6020
Timer #1 (speaker)	21	6021
Timer #2 (8259 IR6)	22	6022
Timer Control	23	6023
Parallel Printer	**30 to 31 hex**	**6030 to 6031 hex**
Printer Data	30	6030
Printer Control Lines	31	6031
8255 PPI Port	**40 to 43 hex**	**6040 to 6043 hex**
Port A (LEDs)	40	6040
Port B (DIP switches)	41	6041
Port C	42	6042
PPI Control	43	6043
8279 LED Display/Keypad	**50 to 51 hex**	**6050 to 6051 hex**
Display/Keypad Data	50	6050
Display/Keypad Control	51	6051
RAMDISK	**60 to 80 hex**	**6060 to 6080 hex**
RAMDISK Low Address	60	6060
RAMDISK High Address	70	6070
RAMDISK Data	80	6080
Analog I/O	**90 to B0 hex**	**6090 to 60B0 hex**
A/D Sample & Hold	90	6090
A/D Select	A0	60A0
D/A select	B0	60B0
8251 Comport 1	**C0 to C1 hex**	**60C0 to 60C1 hex**
COM1 Serial Data	C0	60C0
COM1 Serial Control	C1	60C1
8251 Comport 2	**D0 to D1 hex**	**60D0 to 60D1 hex**
COM2 Serial Data	D0	60D0
COM2 Serial Control	D1	60D1
External Free I/O	**E0 to FF hex**	**60E0 to 60FF hex**

APPENDIX B